

# SPRY 1.4

## デベロッパーガイド



© 2007 Adobe Systems Incorporated. All rights reserved.

Spry framework for Ajax ユーザーガイド Windows® / Macintosh® 版

本マニュアルがエンドユーザー使用許諾契約を含むソフトウェアと共に提供される場合、本マニュアルおよびその中に記載されているソフトウェアは、エンドユーザー使用許諾契約にもとづいて提供されるものであり、当該エンドユーザー使用許諾契約の契約条件に従ってのみ使用または複製することが可能となるものです。当該エンドユーザー使用許諾契約により許可されている場合を除き、本マニュアルのいかなる部分といえども、Adobe Systems Incorporated (アドビ システムズ社) の書面による事前の許可なしに、電子的、機械的、録音、その他いかなる形式・手段であれ、複製、検索システムへの保存、または伝送を行うことはできません。本マニュアルの内容は、エンドユーザー使用許諾契約を含むソフトウェアと共に提供されていない場合であっても、著作権法により保護されていることにご留意ください。

本マニュアルに記載される内容は、あくまでも参照用としてのみ使用されること、また、なんら予告なしに変更されることを条件として、提供されるものであり、従って、当該情報が、アドビ システムズ社による確約として解釈されてはなりません。アドビ システムズ社は、本マニュアルにおけるいかなる誤りまたは不正確な記述に対しても、いかなる義務や責任を負うものではありません。

新しいア트워크を創作するためにテンプレートとして取り込もうとする既存のア트워크または画像は、著作権法により保護されている可能性のあるものであることをご留意ください。保護されているア트워크または画像を新しいア트워크に許可なく取り込んだ場合、著作権者の権利を侵害することがあります。従って、著作権者から必要なすべての許可を必ず取得してください。

例として使用されている会社名は、実在の会社・組織を示すものではありません。

Adobe、Adobe ロゴ、Dreamweaver、および Flash は、アドビ システムズ社の米国ならびに他の国における商標または登録商標です。

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Apple and Mac OS are trademarks of Apple Inc., registered in the United States and other countries. All other trademarks are the property of their respective owners.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. § 2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. § 12.212 or 48 C.F.R. § 227.7202, as applicable. Consistent with 48 C.F.R. § 12.212 or 48 C.F.R. § 227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. アドビ システムズ社は、エンドユーザーである合衆国政府のため、すべての機会均等法 (執行命令 11246 の規定、1974 年 Vietnam Era Veterans Readjustment Assistance Act (38 USC 4212) 402 条および 1973 年 Rehabilitation Act 503 条、ならびに 41 CFR Parts 60-1 乃至 60-60、60-250 および 60-741 の規制を含みます。) を遵守することに同意します。The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

# 目次

## 第 1 章：Spry 1.4 の概要

Spry 1.4 フレームワークについて .....1

## 第 2 章：Spry Widget の操作

Spry Widget について .....2

概要 .....3

アコーディオン Widget の操作 .....4

折りたたみパネル Widget の操作 ..... 13

タブ付きのパネル Widget の操作 ..... 21

メニューバー Widget の操作 ..... 29

テキストフィールド検査 Widget の操作 ..... 39

テキスト領域検査 Widget の操作 ..... 58

選択検査 Widget の操作 ..... 66

チェックボックス検査 Widget の操作 ..... 73

## 第 3 章：Spry XML データセットの使用

Spry XML データセットと動的領域について ..... 80

Spry による動的ページの作成 ..... 97

データの取得と操作 ..... 106

動的領域の使用 ..... 112

## 第 4 章：Spry 効果の使用

Spry 効果について ..... 122

概要 ..... 123

効果の適用 ..... 123

索引 ..... 132

# 第 1 章：Spry 1.4 の概要

Spry 1.4 framework for Ajax は、より高度でおもしろいインタラクティブな Web ページの作成を可能にする、Web デザイナー向けの JavaScript ライブラリです。

## Spry 1.4 フレームワークについて

### Spry 1.4 フレームワークについて

Spry 1.4 framework for Ajax は、豊かなサイト体験が得られる Web ページの作成を可能にする JavaScript ライブラリです。Spry では、HTML コード、CSS コード、および最小限の JavaScript を使用して、XML データを HTML ドキュメントに組み込むことや、アコーディオンやメニューバーなどの仕掛けを作成することや、さまざまなページエレメントに多彩な効果を追加することができます。Spry フレームワークは、HTML、CSS、および JavaScript の基礎的な知識があれば容易にコードを使用できるように設計されています。

Spry フレームワークは、主にプロの Web デザイナーや上級のノンプロ Web デザイナーを対象としています。企業レベルの Web 開発で使用される完全装備型 Web アプリケーションフレームワークとしては設計されていません。ただし、他の企業レベルページと組み合わせることはできます。

Spry 1.4 フレームワークでは、動的ページの作成を可能にする、Widget、XML データセット、および効果という 3 つの主要なコンポーネントを使用できます。Widget は、アコーディオンやタブ付きのパネルなどのページエレメントで、これを使用することでよりおもしろいインタラクティブなページの作成が可能になります。XML データセットを使用すると、従来の Web アプリケーションを使用してデータベースからデータを表示するのと同じように、XML データソースのデータを Web ページ上で表示できるようになります。効果には、フェードやスキッシュなどがあり、それらのモーションをページに追加することによってユーザーの操作性を向上させることができます。Widget 内に XML データを表示したり、Widget に効果を追加したりすることで、静的 HTML を使用するよりもはるかに高度なページを作成できます。以降の章の各項で、Widget、データセット、および効果の使用方法についてそれぞれ詳しく説明します。

Widget、データセット、および効果を組み合わせて使用するサンプルなど、Spry フレームワークの使用法の例については、Adobe Labs の Spry フレームワークのホームページ (<http://labs.adobe.com/technologies/spry/>) を参照してください。このページには、Spry の最新のアップデートもあります。

### Spry 1.4 と Dreamweaver について

Spry 1.4 は、Dreamweaver CS3 で装備しているバージョンの Spry です。そのため、本マニュアルは、Dreamweaver に用意されている Spry アセットを利用するように構成されています。ただし、このバージョンのマニュアルには、Dreamweaver CS3 での Spry ページの作成に役立つビジュアルツールについての説明はありません。Dreamweaver で Spry ツールを使用する方法については、Dreamweaver ヘルプを参照してください。

Spry フレームワークの最新バージョン (Spry 1.5 以降など) のヘルプについては、[www.adobe.com/go/learn\\_spry\\_jp](http://www.adobe.com/go/learn_spry_jp) を参照してください。今後のバージョンの Spry ヘルプは、Dreamweaver に含まれている現在の Spry アセットに対応しない場合があります。Dreamweaver CS3 を使用している場合に、より新しいバージョンの Spry フレームワークを使用するには、Adobe Labs から最新の Spry アセットもダウンロードしてください。

## 第 2 章：Spry Widget の操作

Spry Widget は、HTML、CSS、および JavaScript データを組み合わせることでユーザーインタラクションを可能にするページエレメントです。Spry framework for Ajax は、標準 HTML、CSS、および JavaScript コードで書かれた再利用可能な一連の Widget をサポートします。

### Spry Widget について

#### Spry Widget について

**Spry Widget** は、HTML、CSS、および JavaScript コードを組み合わせることでユーザーインタラクションを可能にするページエレメントです。Spry Widget は次の部分で構成されます。

**Widget 構造** Widget の構成を定義する HTML コードブロック。

**Widget ビヘイビア** ユーザーが開始したイベントに対する Widget の反応をコントロールする JavaScript コード。

**Widget スタイル** Widget の外観を指定する CSS コード。

Spry フレームワークは、標準 HTML、CSS、および JavaScript コードで書かれた再利用可能な一連の Widget をサポートします。これらの Widget は、最も単純なものは HTML と CSS のコードであり、容易に挿入し修正して使用できます。フレームワークのビヘイビアには、ユーザーがさまざまな操作をできるようにする機能が含まれます。これにはたとえば、ページコンテンツの表示と非表示を切り替える、ページの色などの外観を変える、メニュー項目をインタラクティブに使用するなどの機能があります。

Spry フレームワークの各 Widget は、それぞれ固有の CSS および JavaScript (Adobe Labs で入手可能) に関連付けられています。CSS ファイルには、Widget のスタイル設定に必要なすべての設定が含まれています。JavaScript ファイルは、Widget の機能を提供します。ある Widget に関連付けられた CSS および JavaScript ファイルは、その Widget にちなんで名前が付けられています。このため、どのファイルがどの Widget に対応するかを容易に見分けられます。たとえば、アコーディオン Widget に関連付けられたファイルの名前は "SpryAccordion.css" および "SpryAccordion.js" です。

#### Spry Widget のアクセシビリティと JavaScript の適用範囲縮小

Widget の操作性にとって重要なのは、確立された Web ナビゲーション規則に従う場合のアクセシビリティです。ユーザーがマウスを使用していることを想定できないため、Adobe では、現在使用できる Widget のすべての部分にキーボードでアクセスできるようにするための手段を講じています。たとえば、アコーディオン Widget では、上矢印キーと下矢印キーを使用してコンテンツパネルを開くことができます。このような機能を組み込むことをすべての Widget デベロッパーにお勧めします。

エンドユーザーの環境を制御することも不可能です。Adobe では、JavaScript がオフになっている場合でも Widget のすべてのコンテンツが画面に表示されるようにする Widget を開発しています。多くの場合、これはページレイアウトに影響しますが、特に公開 Widget を操作する場合は、Widget のコンテンツが表示されるようにすることの方が重要です。Adobe では、CSS のデフォルト状態で表示が hidden に設定されないようにし、HTML コードが画面外に配置されないようにしています。

#### Spry Widget の開発に関するコーディングのガイドライン

Spry の目的の 1 つに、ユーザーコミュニティが Widget を作成および共有できるようにすることが挙げられます。Adobe は、公開配布のために Widget を作成する際に使用する一連のガイドラインを用意しています。このガイドラインに従って、すべての Widget の基本機能を統一してください。

- 構造には標準 HTML コードを使用します。
- 必要のない限り CSS コードを要求しないでください。

- 機能のために CSS コードを要求する場合は、要件を明確に文書化します。
- 1 行 (可能な場合) の JavaScript を使用して Widget の機能を有効にします。
- デフォルトで、キーボードによる操作のオプションを関数で記述します。
- JavaScript がオフになっている場合でもコンテンツがページに表示されるようにします。
- Widget は自己完結型である必要があります。Widget に必要なものはすべて HTML、JavaScript、および CSS ファイルに提供されます。

これらのガイドラインに従うと、Widget が理解しやすく使いやすいものになり、一貫性によってすべてのユーザーのフレームワークが強化されます。

標準コードを使用することは重要です。一般的なユーザーが知識を習得する必要が少ないためです。また、標準コードを使用すると、Widget を WYSIWYG (What You See Is What You Get) エディタで簡単に使用できます。

CSS コードは一部の Widget で使用し、CSS コードで表示ルールを切り替えることによってコンテンツを表示または非表示にします。これは、CSS を使用する必要がある場合の一例です。CSS コードは明らかにコンテンツを表示または非表示にするためのメカニズムであるため、このような場合は使用可能です。ただし、純粋なスタイル設定である CSS コードは要求しないでください。Widget は、スタイル設定なしで機能する必要があります。CSS ファイルに必要な CSS ルールをコメント付きで文書化します。さらにマニュアルを提供する場合は、その旨も示します。

ほとんどの Widget は、実際の Widget コードの直後にある 1 行の JavaScript コードでアクティブになります。JavaScript パラメータは最小限に抑えるようにしてください。Widget の幅および高さは、他に方法がない場合を除き、JavaScript ではなく CSS コードで設定してください。

キーボードによる操作とアクセシビリティは、ユーザーおよび Spry にとって重要です。ユーザーが一般的なワークフローキー (矢印キー、スペースキー) を使用して Widget のすべての部分にアクセスできるようにキーボードによる操作を記述します。必要に応じてタブ順序などを使用してください。

スクリプティング環境以外の環境では、コンテンツが非表示にならないことが重要です。JavaScript がオフになっている場合に、CSS 表示のオフまたはコンテンツの画面外への配置のためにコンテンツが非表示になることがないようにしてください。

## 概要

### ファイルの準備

Spry Widget を Web ページに追加する前に、該当するファイルをダウンロードしてリンクします。

- 1 Adobe® Labs Web サイトで Spry ZIP ファイルを見つけます。
- 2 Spry ZIP ファイルをハードドライブにダウンロードして解凍します。
- 3 解凍した "Spry" フォルダを開き、"widgets" フォルダを見つけます。このフォルダには、Spry Widget の追加およびスタイル設定に必要なすべてのファイルが格納されています。
- 4 次のいずれかの操作を行って、Widget ファイルを Web サイトに追加します。
  - "widgets" フォルダをコピーして、そのコピーを Web サイトのルートディレクトリにペーストまたはドラッグします。この操作を行うと、Spry でサポートされるすべての Widget に必要なすべてのファイルが準備されます。
  - Web サイトにフォルダ (たとえば、"SpryAssets" という名前のフォルダ) を作成し、"widgets" フォルダを開き、追加する Widget に関連するファイルまたはフォルダのみをコピーします。たとえば、アコーディオン Widget のみを Web ページに追加するには、"accordion" フォルダまたは "SpryAccordion.js" および "SpryAccordion.css" ファイルをコピーします。

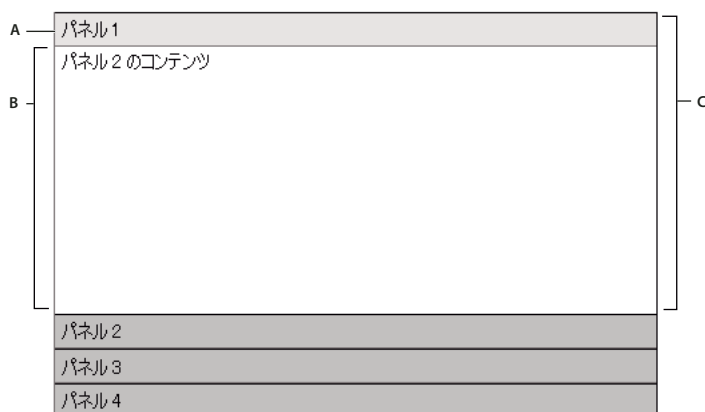
**注意:** 解凍した "Spry" フォルダから元の "widgets" フォルダまたは個別ファイルをドラッグすると、"Spry" フォルダに含まれるデモが正常に動作しません。Web サイトにはドラッグするのではなく、コピー & ペーストしてください。

5 正しい Widget JavaScript および CSS ファイルを Web サイトに含めると、これらのファイルをリンクして Spry Widget をページに追加できるようになります。各 Widget をページに追加する具体的な手順については、それぞれの Widget に関する項を参照してください。

## アコーディオン Widget の操作

### アコーディオン Widget の概要と構造

アコーディオン Widget は、折りたたみ式のパネルのセットです。アコーディオン Widget を使用すると、わずかなスペースに多くのコンテンツを格納できます。ユーザーがパネルのタブをクリックすると、アコーディオンに格納されたコンテンツの表示 / 非表示が切り替わります。アコーディオンの各パネルは、ユーザーがクリックしたタブに応じて拡張および縮小されます。各アコーディオンでは、一度に 1 つのコンテンツパネルのみが開いて表示されます。次にアコーディオン Widget の例を示します。この例では 2 番目のパネルが拡張されています。



A. アコーディオンパネルタブ B. アコーディオンパネルのコンテンツ C. アコーディオンパネル (開いているパネル)

アコーディオン Widget のデフォルトの HTML コードは、すべてのパネルが含まれる外側の div タグ、各パネルの div タグ、各パネルのタグ内のヘッダー div およびコンテンツ div で構成されます。アコーディオン Widget には任意の数のパネルを含めることができます。アコーディオン Widget の HTML コードには、ドキュメントのヘッド内とアコーディオンの HTML コードの後に script タグも含まれます。

ドキュメントのヘッド内の script タグは、アコーディオン Widget に関連するすべての JavaScript 関数を定義します。アコーディオン Widget コードの後の script タグは、アコーディオンをインタラクティブにする JavaScript オブジェクトを作成します。アコーディオン Widget の HTML コードを次に示します。

```
<head>
...
<!--Link the CSS style sheet that styles the accordion-->
<link href="SpryAssets/SpryAccordion.css" rel="stylesheet" type="text/css" />
<!--Link the Spry Accordion JavaScript library-->
<script src="SpryAssets/SpryAccordion.js" type="text/javascript"></script>
</head>
<body>
<!--Create the Accordion widget and assign classes to each element-->
<div id="Accordion1" class="Accordion">
  <div class="AccordionPanel">
    <div class="AccordionPanelTab">Panel 1</div>
    <div class="AccordionPanelContent">
      Panel 1 Content<br/>
      Panel 1 Content<br/>
      Panel 1 Content<br/>
    </div>
  </div>
  <div class="AccordionPanel">
    <div class="AccordionPanelTab">Panel 2</div>
    <div class="AccordionPanelContent">
      Panel 2 Content<br/>
      Panel 2 Content<br/>
      Panel 2 Content<br/>
    </div>
  </div>
  <div class="AccordionPanel">
    <div class="AccordionPanelTab">Panel 3</div>
    <div class="AccordionPanelContent">
      Panel 3 Content<br/>
      Panel 3 Content<br/>
      Panel 3 Content<br/>
    </div>
  </div>
  <div class="AccordionPanel">
    <div class="AccordionPanelTab">Panel 4</div>
    <div class="AccordionPanelContent">
      Panel 4 Content<br/>
      Panel 4 Content<br/>
      Panel 4 Content<br/>
    </div>
  </div>
</div>
<script type="text/javascript">
  var Accordion1 = new Spry.Widget.Accordion("Accordion1");
</script>
</body>
```

このコードでは、new JavaScript 演算子によって、アコーディオン Widget オブジェクトが初期化され、div コンテンツが Accordion1 の ID を使用して静的 HTML コードからインタラクティブなページエレメントに変換されます。Spry.Widget.Accordion メソッドはアコーディオンオブジェクトを作成する Spry フレームワークのコンストラクタです。このオブジェクトの初期化に必要な情報は、ドキュメントのヘッド内でリンクした SpryAccordion.js JavaScript ライブラリに格納されています。

アコーディオン Widget の各 div タグには、CSS クラスが格納されています。このクラスはアコーディオン Widget のスタイルを制御し、対応する "SpryAccordion.css" ファイルに存在します。

アコーディオン Widget の特定の部分の外観を変更するには、HTML コードでその部分に割り当てられたクラス名に対応する CSS コードを編集します。たとえば、アコーディオンのタブの背景色を変更するには、"SpryAccordion.css" ファイルの AccordionPanelTab ルールを編集します。"SpryAccordion.css" ファイルの CSS コードを変更すると、そのファイルにリンクされているすべてのアコーディオンが影響を受けます。

HTML コードに示されているクラスに加えて、アコーディオン Widget には、Widget に関連付けられた特定のデフォルトのビヘイビアもあります。このビヘイビアは Spry フレームワークの組み込み部分で、"SpryAccordion.js" JavaScript ライブラリファイルに格納されています。アコーディオンライブラリには、マウスオーバー、パネルを開くためのタブのクリック、パネルのフォーカス、およびキーボードによる操作に関連するビヘイビアが格納されています。

アコーディオンの外観を変更してこのビヘイビアに関連付けるには、"SpryAccordion.css" ファイルの適切なクラスを編集します。何らかの理由で特定のビヘイビアを削除する場合は、そのビヘイビアに対応する CSS ルールを削除できます。

**注意：**アコーディオンの外観を変更して特定のビヘイビアに関連付けることはできますが、組み込みビヘイビア自体は変更できません。たとえば、"SpryAccordion.css" ファイルの `AccordionFocused` クラスにプロパティが設定されていない場合でも、フォーカスのあるアコーディオンに `AccordionFocused` クラスが適用されます。

### アコーディオン Widget 構造に使用されるさまざまなタグ

上の例では、div タグによって Widget のネストされたタグ構造が作成されます。

```
Container div
  Panel div
    Tab div
      Content div
```

アコーディオン Widget が正常に機能するためには、この 3 レベルの 4 つのコンテナ構造が不可欠です。ただし、この構造は、使用する実際のタグより重要です。Spry で構造 (div タグとは限りません) が読み取られ、それによって Widget が作成されます。3 レベルの 4 つのコンテナ構造が適切に配置されていれば、ブロックレベルの要素を使用して Widget を作成できます。

```
Container div
  Panel div
    H3 tag
    P tag
```

この例の div タグは柔軟で、他のブロックレベルの要素を格納できます。ただし、p タグまたはその他のインラインタグには他のブロックレベルの要素を格納できないため、このタグをアコーディオンのコンテナまたはパネルタグとして使用することはできません。

### アコーディオン Widget の CSS コード

"SpryAccordion.css" ファイルには、アコーディオン Widget のスタイルを設定するルールが含まれています。このルールを編集して、アコーディオンの外観と印象に関するスタイル設定を実行できます。CSS ファイル内のルールの名前は、アコーディオン Widget の HTML コードで指定されたクラスの名前に直接対応します。

**注意：**"SpryAccordion.css" ファイルおよび HTML コードのスタイル関連のクラス名を独自のクラス名に置き換えることができます。CSS コードは Widget の機能には不要であるため、置き換えても Widget の機能には影響しません。

スタイルシートの末尾にあるビヘイビアクラスのデフォルト機能は、JavaScript ライブラリファイル "SpryAccordion.js" で定義されます。デフォルト機能を変更するには、スタイルシートのビヘイビアルールにプロパティおよび値を追加します。

"SpryAccordion.css" ファイルの CSS コードを次に示します。

```
/*Accordion styling classes*/
.Accordion {
  border-left: solid 1px gray;
  border-right: solid 1px black;
  border-bottom: solid 1px gray;
  overflow: hidden;
}
.AccordionPanel {
  margin: 0px;
  padding: 0px;
}
.AccordionPanelTab {
  background-color: #CCCCCC;
  border-top: solid 1px black;
  border-bottom: solid 1px gray;
  margin: 0px;
  padding: 2px;
  cursor: pointer;
}
.AccordionPanelContent {
  overflow: auto;
  margin: 0px;
  padding: 0px;
  height: 200px;
}
.AccordionPanelOpen .AccordionPanelTab {
  background-color: #EEEEEE;
}
.AccordionPanelClosed .AccordionPanelTab {
}
/*Accordion behaviors classes*/
.AccordionPanelTabHover {
  color: #555555;
}
.AccordionPanelOpen .AccordionPanelTabHover {
  color: #555555;
}
}
.AccordionFocused .AccordionPanelTab {
  background-color: #3399FF;
}
}
.AccordionFocused .AccordionPanelOpen .AccordionPanelTab {
  background-color: #33CCFF;
}
}
```

"SpryAccordion.css" ファイルには、特定のルールに対するコードおよび目的を説明するさまざまなコメントが含まれます。詳細については、ファイル内のコメントを参照してください。

## アコーディオン Widget の挿入

1 "SpryAccordion.js" ファイルを検索して、Web サイトに追加します。"SpryAccordion.js" ファイルは、Adobe Labs からダウンロードした Spry ディレクトリにある widgets/accordion ディレクトリ内で検索できます。詳細については、3 ページの「ファイルの準備」を参照してください。

たとえば、Web サイトのルートフォルダ内に "SpryAssets" という名前のフォルダを作成し、このフォルダに "SpryAccordion.js" ファイルを移動します。"SpryAccordion.js" ファイルには、アコーディオン Widget をインタラクティブにするために必要な情報がすべて含まれています。

2 "SpryAccordion.css" ファイルを検索して、Web サイトに追加します。このファイルの追加先として、メインディレクトリである SpryAssets ディレクトリ、または CSS ディレクトリがある場合はそのディレクトリを選択できます。

3 アコーディオン Widget の追加先となる Web ページを開き、このページのヘッドタグに次の script タグを挿入してこのページに "SpryAccordion.js" ファイルをリンクします。

```
<script src="SpryAssets/SpryAccordion.js" type="text/javascript"></script>
```

"SpryAccordion.js" ファイルへのファイルパスが正しいことを確認します。このパスは、Web サイト内のどこにファイルを配置したかによって異なります。

**4** ページのヘッドタグに次の link タグを挿入して、Web ページに "SpryAccordion.css" ファイルをリンクします。

```
<link href="SpryAssets/SpryAccordion.css" rel="stylesheet" type="text/css" />
```

"SpryAccordion.css" ファイルへのファイルパスが正しいことを確認します。このパスは、Web サイト内のどこにファイルを配置したかによって異なります。

**5** ページ上のアコーディオンの表示位置に次の div タグを挿入して、Web ページにアコーディオンを追加します。

```
<div id="Accordion1" class="Accordion">
</div>
```

**6** 次のように <div class="AccordionPanel"> タグを <div id="Accordion1"...> タグ内に挿入して、アコーディオンにパネルを追加します。

```
<div id="Accordion1" class="Accordion">
  <div class="AccordionPanel">
  </div>
  <div class="AccordionPanel">
  </div>
</div>
```

上のコードでは、アコーディオンに 2 つのパネルが追加されます。追加できるパネル数は無制限です。

**7** パネルにタブを追加するには、次のように div class="AccordionPanelTab" タグを各 div class="AccordionPanel" タグ内に挿入します。

```
<div class="AccordionPanel">
  <div class="AccordionPanelTab">Panel 1 Title</div>
</div>
```

**8** パネルにコンテンツ領域を追加するには、次のように div class="AccordionPanelContent" タグを各 div class="AccordionPanel" タグ内に挿入します。

```
<div class="AccordionPanel">
  <div class="AccordionPanelTab">Panel 1 Title</div>
  <div class="AccordionPanelContent">Panel 1 Content</div>
</div>
```

AccordionPanelContent の開始タグと終了タグの間にコンテンツを挿入します。たとえば、前の例では Panel 1 Content です。コンテンツには、HTML フォームエレメントなどの有効な HTML コードを指定できます。

**9** Spry アコーディオンオブジェクトを初期化するには、次のスクリプトブロックをアコーディオン Widget に対応する HTML コードの後に挿入します。

```
<div id="Accordion1" class="Accordion">
. . .
</div>
<script type="text/javascript">
  var Accordion1 = new Spry.Widget.Accordion("Accordion1");
</script>
```

new JavaScript 演算子によって、アコーディオン Widget オブジェクトが初期化され、div コンテンツが Accordion1 の ID を使用して静的 HTML コードからインタラクティブなアコーディオンオブジェクトに変換されます。

Spry.Widget.Accordion メソッドは、アコーディオンオブジェクトを作成する Spry フレームワークのコンストラクタです。このオブジェクトを初期化するために必要な情報は、この手順の開始時にリンクした SpryAccordion.js JavaScript ライブラリに含まれています。

アコーディオンの div タグの ID が、Spry.Widgets.Accordion メソッドで指定した ID パラメータと一致することを確認します。JavaScript の呼び出しが Widget に対応する HTML コードの後に行われることを確認します。

**10** ページを保存します。

コード全体を次に示します。

```
<head>
...
<link href="SpryAssets/SpryAccordion.css" rel="stylesheet" type="text/css" />
<script src="SpryAssets/SpryAccordion.js" type="text/javascript"></script>
</head>
<body>
  <div id="Accordion1" class="Accordion">
    <div class="AccordionPanel">
      <div class="AccordionPanelTab">Panel 1</div>
      <div class="AccordionPanelContent">
        Panel 1 Content<br/>
        Panel 1 Content<br/>
        Panel 1 Content<br/>
      </div>
    </div>
    <div class="AccordionPanel">
      <div class="AccordionPanelTab">Panel 2</div>
      <div class="AccordionPanelContent">
        Panel 2 Content<br/>
        Panel 2 Content<br/>
        Panel 2 Content<br/>
      </div>
    </div>
    <div class="AccordionPanel">
      <div class="AccordionPanelTab">Panel 3</div>
      <div class="AccordionPanelContent">
        Panel 3 Content<br/>
        Panel 3 Content<br/>
        Panel 3 Content<br/>
      </div>
    </div>
    <div class="AccordionPanel">
      <div class="AccordionPanelTab">Panel 4</div>
      <div class="AccordionPanelContent">
        Panel 4 Content<br/>
        Panel 4 Content<br/>
        Panel 4 Content<br/>
      </div>
    </div>
  </div>
  <script type="text/javascript">
    var Accordion1 = new Spry.Widget.Accordion("Accordion1");
  </script>
</body>
```

## アコーディオン Widget へのパネルの追加

❖ `div class="AccordionPanel"` タグ (パネルタブおよびパネルのコンテンツ領域のタグを含む) をアコーディオンのコンテナ `div` タグ内に挿入します。コードを追加する際、必ず終了タグ `</div>` も追加してください。以下に例を挙げます。

```
<div id="Accordion1" class="Accordion">
  <div class="AccordionPanel">
    <div class="AccordionPanelTab"></div>
    <div class="AccordionPanelContent"></div>
  </div>
</div>
```

上のコードでは、アコーディオン Widget に 1 つのパネルが追加されます。追加できるパネル数は無制限です。

## アコーディオン Widget からのパネルの削除

❖ 目的の `div class="AccordionPanel"` タグおよびそのコンテンツまたは子タグをアコーディオンのコンテナ `div` タグから削除します。コードを削除する際、必ず終了タグ `/div` も削除してください。

## キーボードによる操作の有効化

キーボードによる操作で Widget にアクセスできるようにすることは、すべての Widget にとって重要です。キーボードによる操作では、ユーザーは矢印キーおよびカスタムキーを使用して Widget を制御できます。

キーボードによる操作の基礎は `tabIndex` 属性です。この属性は、ドキュメント内を移動する方法をブラウザに指示します。

❖ アコーディオンでキーボードによる操作を有効にするには、次のように `TabIndex` 値をアコーディオンのコンテンツタグに追加します。

```
<div id="Acc1" class="Accordion" tabIndex="0">
```

`tabIndex` 属性の値がゼロ (0) の場合、ブラウザで順序が決定されます。`tabIndex` 属性の値が正の整数の場合、その順序値が使用されます。

**注意：** `div` タグでの `tabIndex` の使用は XHTML 1.0 に準拠していません。

キーボードによる操作用のカスタムキーを設定することもできます。カスタムキーは、アコーディオンコンストラクタスクリプトのパラメータとして設定されます。

```
<script type="text/javascript">
var acc3= new Spry.Widget.Accordion("Acc3", { nextPanelKeyCode: 78 /* n key */, previousPanelKeyCode: 80
/* p key */ });
</script>
```

## 最初に開くパネルの設定

アコーディオン Widget を含むページがブラウザにロードされるときに特定のパネルが開くように設定できます。

❖ 次のように、コンストラクタで `defaultPanel` オプションを設定します。

```
<script type="text/javascript">
var acc8 = new Spry.Widget.Accordion("Accordion1", { defaultPanel: 2 });
</script>
```

**注意：** アコーディオンパネルではゼロからカウントを開始するシステムが使用されるため、値を 2 に設定すると 3 番目のパネルが開きます。

## プログラムによるパネルの表示

JavaScript 関数を使用して、さまざまなパネルをプログラムで開くことができます。たとえば、クリックすると特定のアコーディオンパネルが開かれるボタンをページに配置できます。

❖ 次の JavaScript 関数を使用してアコーディオンパネルを開きます。

```
<input type="button" onclick="acc10.openFirstPanel()" >open first panel</input>
<input type="button" onclick="acc10.openNextPanel()" >open next panel</input>
<input type="button" onclick="acc10.openPreviousPanel()" >open previous panel</input>
<input type="button" onclick="acc10.openLastPanel()" >open last panel</input>
<script type="text/javascript">
var acc10 = new Spry.Widget.Accordion("Accordion1");
</script>
```

## アコーディオン Widget のカスタマイズ

"SpryAccordion.css" ファイルには、アコーディオン Widget のデフォルトのスタイル設定が用意されています。ただし、該当する CSS を変更することによって簡単に Widget をカスタマイズできます。"SpryAccordion.css" ファイルの CSS ルールでは、アコーディオンの HTML コードの関連するエレメントと同じクラス名が使用されるため、アコーディオン Widget のさまざまなセクションに対応する CSS ルールを簡単に把握できます。さらに、"SpryAccordion.css" ファイルには、Widget に関連するビヘイビア (マウスオーバーやクリックのビヘイビアなど) のクラス名が含まれています。

"SpryAccordion.css" ファイルは、カスタマイズを開始する前に Web サイトに配置しておく必要があります。詳細については、3 ページの「ファイルの準備」を参照してください。

**注意：** Internet Explorer バージョン 6 以前では、兄弟および子コンテキストセレクタ (`.AccordionPanel + .AccordionPanel` や `.Accordion > .AccordionPanel` など) はサポートされていません。

### アコーディオン Widget のスタイル設定 (一般的な方法)

アコーディオン Widget コンテナ全体のプロパティを設定するか、Widget の各コンポーネントのプロパティを個別に設定します。

- 1 "SpryAccordion.css" ファイルを開きます。
- 2 アコーディオンの変更する部分の CSS ルールを見つけます。たとえば、アコーディオンのタブの背景色を変更するには、"SpryAccordion.css" ファイルの AccordionPanelTab ルールを編集します。
- 3 CSS を変更してファイルを保存します。

"SpryAccordion.css" ファイルおよび HTML コードのスタイル関連のクラス名を独自のクラス名に置き換えることができます。置き換えても Widget の機能には影響しません。

"SpryAccordion.css" ファイルには、特定のルールに対するコードおよび目的を説明するさまざまなコメントが含まれます。詳細については、ファイル内のコメントを参照してください。

### アコーディオン Widget のテキストのスタイル設定

アコーディオン Widget コンテナ全体のプロパティを設定するか、Widget の各コンポーネントのプロパティを個別に設定します。

❖ アコーディオン Widget のテキストスタイルを変更するには、次の表を使用して "SpryAccordion.css" ファイルで適切な CSS ルールを見つけ、独自のテキストスタイルプロパティおよび値を追加します。

| 変更するテキスト                             | 関連する CSS ルール                   | 追加するプロパティおよび値の例                 |
|--------------------------------------|--------------------------------|---------------------------------|
| アコーディオン全体 (タブとコンテンツツバネルの両方を含む) のテキスト | .Accordion または .AccordionPanel | font: Arial; font-size: medium; |
| アコーディオンパネルタブのテキストのみ                  | .AccordionPanelTab             | font: Arial; font-size: medium; |
| アコーディオンコンテンツツバネルのテキストのみ              | .AccordionPanelContent         | font: Arial; font-size: medium; |

### アコーディオン Widget の背景色の変更

❖ 次の表を使用して "SpryAccordion.css" ファイルで適切な CSS ルールを見つけ、背景色プロパティおよび値を追加または変更します。

| 変更する Widget の部分           | 関連する CSS ルール                                | 追加または変更するプロパティおよび値の例                    |
|---------------------------|---|---|
| アコーディオンパネルタブの背景色          | .AccordionPanelTab                          | background-color: #CCCCCC; (これが初期設定値です) |
| アコーディオンコンテンツツバネルの背景色      | .AccordionPanelContent                      | background-color: #CCCCCC;              |
| 開いたアコーディオンパネルの背景色         | .AccordionPanelOpen .AccordionPanelTab      | background-color: #EEEEEE; (これが初期設定値です) |
| マウスポインタがあるときのパネルタブの背景色    | .AccordionPanelTabHover                     | color: #555555; (これが初期設定値です)            |
| マウスポインタがあるときの開いたパネルタブの背景色 | .AccordionPanelOpen .AccordionPanelTabHover | color: #555555; (これが初期設定値です)            |

### アコーディオンの幅の固定

初期設定では、アコーディオン Widget は使用可能なスペースいっぱいに拡張されます。アコーディオン Widget の幅を制限するには、アコーディオンコンテナの幅プロパティを設定します。

- 1 "SpryAccordion.css" ファイルで .Accordion CSS ルールを見つけます。このルールは、アコーディオン Widget のメインコンテナエレメントのプロパティを定義します。
- 2 このルールに、width: 300px; のように幅プロパティと値を追加します。

### アコーディオンパネルの高さの変更

デフォルトでは、アコーディオン Widget パネルの高さは 200 ピクセルに設定されています。パネルの高さを変更するには、.AccordionPanelContent ルールの高さプロパティを変更します。

- 1 "SpryAccordion.css" ファイルで .AccordionPanelContent CSS ルールを見つけます。
- 2 高さプロパティを目的の寸法に変更します。

**注意：** この値は常に、アコーディオンパネルの各サイズが同じになるように設定してください。

### アコーディオンパネルのビヘイビアの変更

アコーディオン Widget には、いくつかの定義済みビヘイビアがあります。これらのビヘイビアは、特定のイベントが発生したときの CSS クラスの変更で構成されます。たとえば、マウスポインタがアコーディオンパネルタブの上に置かれると、AccordionPanelTabHover クラスが Widget に適用されます。このビヘイビアは、リンクの a:hover に似ています。ビヘイビアは、デフォルトでは空白です。変更するには、ルールにプロパティおよび値を追加します。

- 1 "SpryAccordion.css" ファイルを開き、変更するアコーディオンビヘイビアの CSS ルールを見つけます。アコーディオン Widget には、ビヘイビアの 4 つの組み込みルールがあります。

| ビヘイビア                   | 説明                               |
|-------------------------|----------------------------------|
| .AccordionPanelTabHover | マウスポインタをパネルタブの上に置くとアクティブになります。   |
| .AccordionPanelOpen     | パネルタブが開くとアクティブになります。             |
| .AccordionPanelClosed   | パネルが閉じるとアクティブになります。              |
| .AccordionFocused       | アコーディオン全体にフォーカスが置かれるとアクティブになります。 |

例については、Adobe Labs からダウンロードした Spry ディレクトリの samples ディレクトリにあるアコーディオンのサンプルファイルを参照してください。詳細については、3 ページの「ファイルの準備」を参照してください。

- 2 有効にするビヘイビアの CSS ルールを追加します。

**注意：** ビヘイビアは Spry フレームワークの一部としてハードコーディングされているため、"SpryAccordion.css" ファイルのビヘイビア関連のクラス名を独自のクラス名に置き換えることはできません。デフォルトのクラスを独自のクラス名で上書きするには、次の例に示すように、新しい値をパラメータとしてアコーディオンコンストラクタに送ります。

```
<script type="text/javascript">
  var acc2 = new Spry.Widget.Accordion("Acc2", { hoverClass: "hover", openClass: "open", closedClass:
"closed", focusedClass: "focused" });
</script>
```

### パネルアニメーションのオフ

デフォルトでは、アコーディオンパネルはアニメーションを使用して滑らかに開閉します。ただし、パネルがすぐに開閉するように、このアニメーションをオフにすることができます。

- ❖ アニメーションをオフにするには、次のようにパラメータをアコーディオンコンストラクタに送ります。

```
<script type="text/javascript">
  var acc5 = new Spry.Widget.Accordion("Acc5", { enableAnimation: false });
</script>
```

### パネルアニメーションのタイミングの設定

パネルが開くときにかかる時間を変更できます。時間はミリ秒単位で設定します (1000 = 1 秒)。デフォルトでは、500 ミリ秒が使用されます。

- ❖ duration オプションをコンストラクタに追加します。

```
<script type="text/javascript">
    var acc9 = new Spry.Widget.Accordion("Acc9", { duration: 100 });
</script>
```

### 変更可能なパネルの高さの設定

デフォルトでは、アニメーションが有効になっている場合、すべてのアコーディオンコンテンツパネルで強制的に同じ高さが使用されます。この高さは、CSS またはパネルのコンテンツの高さで決定されるアコーディオンの最初に開くパネルの高さから取得されます。

アコーディオンでは、高さが可変のパネルもサポートされます。このサポートをオンにするには、`useFixedPanelHeights: false` オプションをアコーディオンのコンストラクタに渡します。

❖ 次のように、`useFixedPanelHeights: false` オプションを渡します。

```
<script type="text/javascript">
    var acc7 = new Spry.Widget.Accordion("Acc7", { useFixedPanelHeights: false });
</script>
```

例については、Adobe Labs からダウンロードした Spry ディレクトリの `samples` ディレクトリにあるアコーディオンのサンプルファイルを参照してください。詳細については、3 ページの「ファイルの準備」を参照してください。

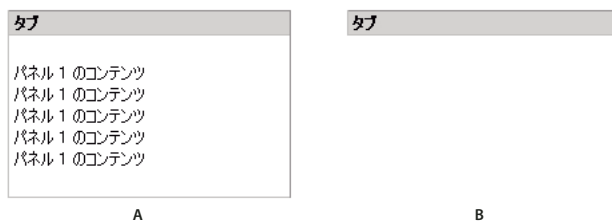
Spry でパネルの高さが CSS コードではなく JavaScript の設定値に設定されるようにするには、コンテンツパネルの高さをプログラムで設定する `fixedPanelHeight` オプションを渡します。このオプションではピクセルが使用されます。

```
<script type="text/javascript">
    var acc7 = new Spry.Widget.Accordion("Acc7", { fixedPanelHeight: "300px" });
</script>
```

## 折りたたみパネル Widget の操作

### 折りたたみパネル Widget の概要と構造

折りたたみパネル Widget は、わずかなスペースにコンテンツを格納できるパネルです。折りたたみパネルに格納されたコンテンツは、ユーザーが Widget のタブをクリックすると表示 / 非表示が切り替わります。次に示すのは、折りたたみパネル Widget が展開された例と縮小された例です。



A. 展開 B. 縮小

折りたたみパネル Widget の HTML コードは、外側の `div` タグと、このタグに含まれるコンテンツ `div` タグおよびコンテンツ `div` タグで構成されます。折りたたみパネル Widget の HTML コードには、ドキュメントのヘッド内と折りたたみパネルの HTML コードの後に `script` タグも含まれます。

ドキュメントのヘッド内の `script` タグは、折りたたみパネル Widget に関連するすべての JavaScript 関数を定義します。折りたたみパネル Widget のコードの後の `script` タグは、折りたたみパネルをインタラクティブにする JavaScript オブジェクトを作成します。折りたたみパネル Widget の HTML コードを次に示します。

```
<head>
...
<!--Link the CSS style sheet that styles the Collapsible Panel-->
<link href="SpryAssets/SpryCollapsiblePanel.css" rel="stylesheet"
      type="text/css" />
<!--Link the Spry Collapsible Panel JavaScript library-->
<script src="SpryAssets/SpryCollapsiblePanel.js" type="text/javascript"></script>
</head>
<body>
<!--Create the Collapsible Panel widget and assign classes to each element-->
<div id="CollapsiblePanel1" class="CollapsiblePanel">
  <div class="CollapsiblePanelTab">Tab</div>
  <div class="CollapsiblePanelContent">Content</div>
</div>
<!--Initialize the Collapsible Panel widget object-->
<script type="text/javascript">
  var CollapsiblePanel1 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel1");
</script>
</body>
```

このコードでは、new JavaScript 演算子によって、折りたたみパネル Widget オブジェクトが初期化され、div コンテンツが CollapsiblePanel1 の ID を使用して静的 HTML コードからインタラクティブなページエレメントに変換されます。Spry.Widget.CollapsiblePanel メソッドは折りたたみパネルオブジェクトを作成する Spry フレームワークのコンストラクタです。このオブジェクトを初期化するために必要な情報は、ドキュメントのヘッド内でリンクした SpryCollapsiblePanel.js JavaScript ライブラリに含まれています。

折りたたみパネル Widget の各エレメントには、CSS クラスが格納されています。このクラスは折りたたみパネル Widget のスタイルを制御し、対応する "SpryCollapsiblePanel.css" ファイルに存在します。

折りたたみパネル Widget の特定の部分の外観を変更するには、HTML コードでその部分に割り当てられたクラス名に対応する CSS コードを編集します。たとえば、折りたたみパネルのタブの背景色を変更するには、"SpryCollapsiblePanel.css" ファイルの CollapsiblePanelTab ルールを編集します。"SpryCollapsiblePanel.css" ファイルの CSS コードを変更すると、そのファイルにリンクされているすべての折りたたみパネルに影響を受けます。

HTML コードに示されているクラスに加えて、折りたたみパネル Widget には、Widget に関連付けられた特定のデフォルトのビヘイビアもあります。このビヘイビアは Spry フレームワークの組み込み部分で、"SpryCollapsiblePanel.js" JavaScript ライブラリファイルに格納されています。折りたたみパネルライブラリには、マウスオーバー、パネルを開閉するためのクリック、パネルのフォーカス、およびキーボードによる操作に関連するビヘイビアが格納されています。

折りたたみパネルの外観を変更してこのビヘイビアに関連付けるには、"SpryCollapsiblePanel.css" ファイルの適切なクラスを編集します。何らかの理由で特定のビヘイビアを削除する場合は、そのビヘイビアに対応する CSS ルールを削除できます。

**注意：**折りたたみパネルの外観を変更して特定のビヘイビアに関連付けることはできますが、組み込みビヘイビア自体は変更できません。たとえば、"SpryCollapsiblePanel.css" ファイルの CollapsiblePanelFocused クラスにプロパティが設定されていない場合でも、現在開いているパネルに CollapsiblePanelFocused クラスが適用されます。

### 折りたたみパネル Widget 構造に使用されるさまざまなタグ

上の例では、div タグによって Widget のネストされたタグ構造が作成されます。

```
Container div
  Tab div
  Content div
```

折りたたみパネル Widget が正常に機能するためには、この 2 レベルの 3 つのコンテナ構造が不可欠です。ただし、この構造は、使用する実際のタグより重要です。Spry で構造 (div タグとは限りません) が読み取られ、それに従って Widget が作成されます。2 レベルの 3 つのコンテナ構造が適切に配置されていれば、ブロックレベルのエレメントを使用して Widget を作成できます。

```
Container div
  H3 tag
  P tag
```

この例の div タグは柔軟で、他のブロックレベルの要素を格納できます。ただし、p タグまたはその他のインラインタグには他のブロックレベルの要素を格納できないため、このタグを折りたたみパネルのコンテナまたはパネルタグとして使用することはできません。

## 折りたたみパネル Widget の CSS コード

"SpryCollapsiblePanel.css" ファイルには、折りたたみパネル Widget のスタイルを設定するルールが含まれています。このルールを編集して、折りたたみパネルの外観と印象に関するスタイル設定を実行できます。CSS ファイルのルールの名前は、折りたたみパネル Widget の HTML コードで指定されたクラスの名前に直接対応します。

**注意：**"SpryCollapsiblePanel.css" ファイルおよび HTML コードのスタイル関連のクラス名を独自のクラス名に置き換えることができます。CSS コードは Widget の機能には不要であるため、置き換えても Widget の機能には影響しません。

スタイルシートの末尾にあるビヘイビアクラスのデフォルト機能は、"SpryCollapsiblePanel.js" JavaScript ライブラリファイルで定義されます。デフォルト機能を変更するには、スタイルシートのビヘイビアルールにプロパティおよび値を追加します。

"SpryCollapsiblePanel.css" ファイルの CSS コードを次に示します。

```
/*Collapsible Panel styling classes*/
.CollapsiblePanel {
    margin: 0px;
    padding: 0px;
    border-left: solid 1px #CCC;
    border-right: solid 1px #999;
}
.CollapsiblePanelTab {
    font: bold 0.7em sans-serif;
    background-color: #DDD;
    border-top: solid 1px #999;
    border-bottom: solid 1px #CCC;
    margin: 0px;
    padding: 2px;
    cursor: pointer;
    -moz-user-select: none;
    -khtml-user-select: none;
}
.CollapsiblePanelContent {
    margin: 0px;
    padding: 0px;
    border-bottom: solid 1px #CCC;
}
.CollapsiblePanelTab a {
    color: black;
    text-decoration: none;
}
.CollapsiblePanelOpen .CollapsiblePanelTab {
    background-color: #EEE;
}
.CollapsiblePanelTabHover, .CollapsiblePanelOpen .CollapsiblePanelTabHover {
background-color: #CCC;
}
.CollapsiblePanelFocused .CollapsiblePanelTab {
    background-color: #3399FF;
}
```

"SpryCollapsiblePanel.css" ファイルには、特定のルールに対するコードおよび目的を説明するさまざまなコメントが含まれます。詳細については、ファイル内のコメントを参照してください。

## 折りたたみパネル Widget の挿入

1 "SpryCollapsiblePanel.js" ファイルを検索して、Web サイトに追加します。"SpryCollapsiblePanel.js" ファイルは、Adobe Labs からダウンロードした Spry ディレクトリにある widgets/collapsiblepanel ディレクトリ内で検索できます。詳細については、3 ページの「ファイルの準備」を参照してください。

たとえば、Web サイトのルートフォルダ内に "**SpryAssets**" という名前のフォルダを作成し、このフォルダに "**SpryCollapsiblePanel.js**" ファイルを移動します。"**SpryCollapsiblePanel.js**" ファイルには、折りたたみパネル Widget をインタラクティブにするために必要な情報がすべて含まれています。

**2** "**SpryCollapsiblePanel.css**" ファイルを検索して、Web サイトに追加します。このファイルの追加先として、メインディレクトリ、または CSS ディレクトリがある場合はそのディレクトリを選択できます。

**3** 折りたたみパネル Widget の追加先となる Web ページを開き、このページのヘッドタグに次の script タグを挿入してこのページに "**SpryCollapsiblePanel.js**" ファイルをリンクします。

```
<script src="SpryAssets/SpryCollapsiblePanel.js" type="text/javascript"></script>
```

"**SpryCollapsiblePanel.js**" ファイルへのファイルパスが正しいことを確認します。このパスは、Web サイト内のどこにファイルを配置したかによって異なります。

**4** ページのヘッドタグに次の link タグを挿入して、Web ページに "**SpryCollapsiblePanel.css**" ファイルをリンクします。

```
<link href="SpryAssets/SpryCollapsiblePanel.css" rel="stylesheet" type="text/css" />
```

"**SpryCollapsiblePanel.css**" ファイルへのファイルパスが正しいことを確認します。このパスは、Web サイト内のどこにファイルを配置したかによって異なります。

**5** 折りたたみパネルを Web ページに追加するには、ページ上の折りたたみパネルの表示位置に次の div タグを挿入します。

```
<div id="CollapsiblePanel1" class="CollapsiblePanel">
</div>
```

**6** 折りたたみパネルにタブを追加するには、次のように div class="CollapsiblePanelTab" タグを div class="CollapsiblePanel" タグ内に挿入します。

```
<div id="CollapsiblePanel1" class="CollapsiblePanel">
  <div class="CollapsiblePanelTab">Tab</div>
</div>
```

**7** パネルにコンテンツ領域を追加するには、次のように div class="CollapsiblePanelContent" タグを div class="CollapsiblePanel" タグ内に挿入します。

```
<div class="AccordionPanel">
  <div class="CollapsiblePanelTab">Tab</div>
  <div class="CollapsiblePanelContent">Content</div>
</div>
```

CollapsiblePanelContent の開始タグと終了タグの間にコンテンツを挿入します。たとえば、前の例では Content です。コンテンツには、HTML フォームエレメントなどの有効な HTML コードを指定できます。

**8** Spry の折りたたみパネルオブジェクトを初期化するには、折りたたみパネル Widget に対応する HTML コードの後に次のスクリプトブロックを挿入します。

```
<div id="CollapsiblePanel1" class="CollapsiblePanel">
  . . .
</div>
<script type="text/javascript">
  var acc1 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel1");
</script>
```

new JavaScript 演算子によって、折りたたみパネル Widget オブジェクトが初期化され、div コンテンツが CollapsiblePanel1 の ID を使用して静的 HTML コードからインタラクティブな折りたたみパネルオブジェクトに変換されます。Spry.Widget.CollapsiblePanel メソッドは、折りたたみパネルオブジェクトを作成する Spry フレームワークのコンストラクタです。このオブジェクトを初期化するために必要な情報は、この手順の開始時にリンクした SpryCollapsiblePanel.js JavaScript ライブラリに含まれています。

折りたたみパネルの div タグの ID が、Spry.Widgets.CollapsiblePanel メソッドで指定した ID パラメータと一致することを確認します。JavaScript の呼び出しが Widget に対応する HTML コードの後に行われることを確認します。

**9** ページを保存します。

コード全体を次に示します。

```
<head>
...
<link href="SpryAssets/SpryCollapsiblePanel.css" rel="stylesheet" type="text/css" />
<script src="SpryAssets/SpryCollapsiblePanel.js" type="text/javascript"></script>
</head>
<body>
  <div id="CollapsiblePanel1" class="CollapsiblePanel">
    <div class="CollapsiblePanelTab">Tab</div>
    <div class="CollapsiblePanelContent">Content</div>
  </div>
<script type="text/javascript">
  var CollapsiblePanel1 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel1");
</script>
</body>
```

## キーボードによる操作の有効化

キーボードによる操作で Widget にアクセスできるようにすることは、すべての Widget にとって重要です。キーボードによる操作では、ユーザーはスペースキーまたは Enter キーを使用して Widget を制御できます。

キーボードによる操作の基礎は `tabIndex` 属性です。この属性は、タブを使用してドキュメント内を移動する方法をブラウザに指示します。

❖ 折りたたみパネルでキーボードによる操作を有効にするには、次のように `TabIndex` 値を折りたたみパネルのタブタグに追加します。

```
<div id="CollapsiblePanel1" class="CollapsiblePanel">
  <div class="CollapsiblePanelTab" tabIndex="0">Tab</div>
  <div class="CollapsiblePanelContent">Content</div>
</div>
```

`tabIndex` 属性の値がゼロ (0) の場合、ブラウザで順序が決定されます。`tabIndex` 属性の値が正の整数の場合、その順序値が使用されます。

キーボードによる操作を有効にするには、タブコンテンツを `a` タグで囲みます。

**注意：** `div` タグでの `tabIndex` の使用は XHTML 1.0 に準拠していません。

## パネルのデフォルト状態の設定

デフォルトでは、Web ページがブラウザにロードされる時、折りたたみパネル Widget は開いています。ただし、ページがロードされる時にパネルを閉じた状態にする場合は、パネルの状態を変更できます。

❖ 次のように、コンストラクタで `contentIsOpen` オプションを設定します。

```
<script type="text/javascript">
  var CollapsiblePanel1 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel1", { contentIsOpen: false
});
</script>
```

`isOpen` 関数を使用してパネルの状態を確認することもできます。次のコードを実行すると、パネルが開いている場合は `true`、開いていない場合は `false` が返されます。

```
<script type="text/javascript">
function  getStatus(){
var xx = CollapsiblePanel1.isOpen();
document.form1.textfield.value = xx
}
</script>
</head>
<body>
<div id="CollapsiblePanel1" class="CollapsiblePanel">
  <div class="CollapsiblePanelTab" tabIndex="0" onclick="getStatus();">Tab</div>
  <div class="CollapsiblePanelContent">Content</div>
</div>
<form id="form1" name="form1" method="post" action="">
  <input name="textfield" type="text" id="textfield" value="a" size="50" />
</form>
<script type="text/javascript">
<!--
var CollapsiblePanel1 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel1");
//-->
</script>
```

## プログラムによるパネルの表示

JavaScript 関数を使用して、折りたたみパネル Widget をプログラムで開閉できます。たとえば、クリックすると折りたたみパネルが開かれるボタンをページに配置できます。

❖ 次の関数を使用して折りたたみパネルを開閉します。

```
<input type="button" onclick="CollapsiblePanel1.open();" >open panel</input>
<input type="button" onclick="CollapsiblePanel1.close();" >close panel</input>
<script type="text/javascript">
  var CollapsiblePanel1 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel1");
</script>
```

## 折りたたみパネル Widget のカスタマイズ

"SpryCollapsiblePanel.css" ファイルには、折りたたみパネル Widget のデフォルトのスタイル設定が用意されています。ただし、該当する CSS ルールを変更することによって、Widget を簡単にカスタマイズできます。

"SpryCollapsiblePanel.css" ファイルの CSS ルールでは、折りたたみパネルの HTML コードの関連するエレメントと同じクラス名が使用されるため、折りたたみパネル Widget のさまざまなセクションに対応する CSS ルールを簡単に把握できます。さらに、"SpryCollapsiblePanel.css" ファイルには、Widget に関連するビヘイビア (マウスオーバーやクリックのビヘイビアなど) のクラス名が含まれています。

"SpryCollapsiblePanel.css" ファイルは、カスタマイズを開始する前に Web サイトに配置しておく必要があります。詳細については、3 ページの「ファイルの準備」を参照してください。

**注意:** Internet Explorer バージョン 6 以前では、兄弟および子コンテキストセレクタ (.CollapsiblePanel + .CollapsiblePanel や .CollapsiblePanel > .CollapsiblePanel など) はサポートされていません。

### 折りたたみパネル Widget のスタイル設定 (一般的な方法)

折りたたみパネル Widget のコンテナ全体のプロパティを設定するか、Widget の各コンポーネントのプロパティを個別に設定します。

- 1 "SpryCollapsiblePanel.css" ファイルを開きます。
- 2 折りたたみパネルの変更する部分の CSS ルールを見つけます。たとえば、折りたたみパネルのタブの背景色を変更するには、"SpryCollapsiblePanel.css" ファイルの CollapsiblePanelTab ルールを編集します。
- 3 CSS ルールを変更してファイルを保存します。

"SpryCollapsiblePanel.css" ファイルおよび HTML コードのスタイル関連のクラス名を独自のクラス名に置き換えることができます。置き換えても Widget の機能には影響しません。

"SpryCollapsiblePanel.css" ファイルには、特定のルールに対するコードおよび目的を説明するさまざまなコメントが含まれます。詳細については、ファイル内のコメントを参照してください。

### 折りたたみパネル Widget のテキストのスタイル設定

折りたたみパネル Widget のコンテナ全体のプロパティを設定するか、Widget の各コンポーネントのプロパティを個別に設定します。

❖ 折りたたみパネル Widget のテキストフォーマットを変更するには、次の表を使用して適切な CSS ルールを見つけ、独自のテキストスタイルプロパティおよび値を追加します。

| 変更するスタイル        | 関連する CSS ルール             | 追加または変更するプロパティおよび値の例                      |
|-----------------|--------------------------|---|
| 折りたたみパネル全体のテキスト | .CollapsiblePanel        | font: Arial; font-size:medium;            |
| パネルタブのみのテキスト    | .CollapsiblePanelTab     | font: bold 0.7em sans-serif; (これが初期設定値です) |
| コンテンツパネルのみのテキスト | .CollapsiblePanelContent | font: Arial; font-size:medium;            |

### 折りたたみパネル Widget の背景色の変更

❖ 次の表を使用して適切な CSS ルールを見つけ、背景色プロパティおよび値を追加または変更します。

| 変更する色                        | 関連する CSS ルール   | 追加または変更するプロパティおよび値の例                 |
|------------------------------|--|--------------------------------------|
| パネルタブの背景色                    | .CollapsiblePanelTab   | background-color: #DDD; (これが初期設定値です) |
| コンテンツパネルの背景色                 | .CollapsiblePanelContent   | background-color: #DDD;              |
| パネルが開いているときのタブの背景色           | .CollapsiblePanelOpen<br>.CollapsiblePanelTab                                    | background-color: #EEE; (これが初期設定値です) |
| マウスポインタが上にあるときの、開いたパネルタブの背景色 | .CollapsiblePanelTabHover、<br>.CollapsiblePanelOpen<br>.CollapsiblePanelTabHover | background-color: #CCC; (これが初期設定値です) |

### 折りたたみパネルの幅の固定

初期設定では、折りたたみパネル Widget は使用可能なスペースいっぱいに拡張されます。折りたたみパネル Widget の幅を制限するには、折りたたみパネルのコンテナの幅プロパティを設定します。

- 1 "SpryCollapsiblePanel.css" ファイルで CollapsiblePanel CSS ルールを見つけます。このルールは、折りたたみパネル Widget のメインコンテナエレメントのプロパティを定義します。
- 2 このルールに、width: 300px; のように幅プロパティと値を追加します。

### 折りたたみパネルの高さの変更

折りたたみパネルの高さを設定するには、CollapsiblePanelContent ルールまたは CollapsiblePanel ルールに高さプロパティを追加します。

❖ "SpryCollapsiblePanel.css" ファイルの CollapsiblePanelContent ルールに、height: 300px; のように高さプロパティと値を追加します。

**注意：** CollapsiblePanel ルールの高さを設定すると、コンテンツパネルのサイズとは無関係に、Widget 全体の高さが設定されます。

## 折りたたみパネルのビヘイビアの変更

折りたたみパネル Widget には、いくつかの定義済みビヘイビアがあります。これらのビヘイビアは、特定のイベントが発生したときの CSS クラスの変更で構成されます。たとえば、マウスポインタが折りたたみパネルタブの上に置かれると、CollapsiblePanelTabHover クラスが Widget に適用されます。このビヘイビアは、リンクの a: hover に似ています。ビヘイビアは、デフォルトでは空白です。変更するには、ルールにプロパティおよび値を追加します。

1 "SpryCollapsiblePanel.css" ファイルを開き、変更する折りたたみパネルビヘイビアの CSS ルールを見つけます。折りたたみパネル Widget には、ビヘイビアの 4 つの組み込みルールがあります。

| ビヘイビア                     | 説明   |
|---------------------------|--|
| .CollapsiblePanelTabHover | マウスが Widget のタブエレメントの上に置かれると、このクラスが追加されます。マウスがタブから離れると、このクラスは自動的に削除されます。 |
| .CollapsiblePanelOpen     | パネルのコンテンツ領域を表示すると、Widget の最上位エレメントにこのクラスが追加されます。                         |
| .CollapsiblePanelClosed   | パネルのコンテンツ領域を閉じると、Widget の最上位エレメントにこのクラスが追加されます。                          |
| .CollapsiblePanelFocused  | Widget にキーボードフォーカスが置かれると、Widget の最上位エレメントにこのクラスが追加されます。                  |

例については、Adobe Labs からダウンロードした Spry ディレクトリの samples ディレクトリにある折りたたみパネルのサンプルファイルを参照してください。詳細については、3 ページの「ファイルの準備」を参照してください。

2 有効にするビヘイビアの CSS ルールを追加します。

**注意:** ビヘイビアは Spry フレームワークの一部としてハードコーディングされているため、"SpryCollapsiblePanel.css" ファイルのビヘイビア関連のクラス名を独自のクラス名に置き換えることはできません。デフォルトのクラスを独自のクラス名で上書きするには、新しい値をパラメータとして折りたたみパネルコンストラクタに送ります。

```
<script type="text/javascript">
    var CP2 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel2", { hoverClass: "hover", openClass:
"open", closedClass: "closed", focusedClass: "focused" });
</script>
```

### パネルアニメーションのオフ

デフォルトでは、折りたたみパネルはアニメーションを使用して滑らかに開閉します。

❖ パネルがすぐに開閉するようにアニメーションをオフにするには、次のようにパラメータを折りたたみパネルコンストラクタに送ります。

```
<script type="text/javascript">
    var CP5 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel5", { enableAnimation: false });
</script>
```

### パネルアニメーションのタイミングの設定

パネルが開くときにかかる時間を変更できます。時間はミリ秒単位で設定します (1000 = 1 秒)。デフォルトでは、500 ミリ秒が使用されます。

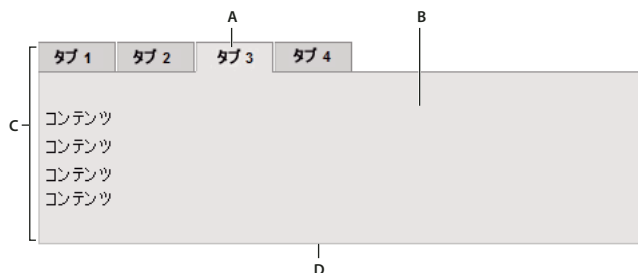
❖ duration オプションをコンストラクタに追加します。

```
<script type="text/javascript">
    var CP9 = new Spry.Widget.CollapsiblePanel("CollapsiblePanel9", { duration: 100 });
</script>
```

## タブ付きのパネル Widget の操作

### タブ付きのパネル Widget の概要と構造

タブ付きのパネル Widget は、わずかなスペースにコンテンツを格納できるパネルのセットです。ユーザーがアクセスしたいパネルのタブをクリックすると、タブ付きのパネルに格納されたコンテンツの表示 / 非表示が切り替わります。ユーザーが別のタブをクリックすると、それによって Widget のパネルが開きます。タブ付きのパネル Widget では、一度に 1 つのコンテンツパネルのみが開きます。次にタブ付きのパネル Widget の例を示します。この例では、3 番目のパネルが開いています。



A. タブ B. コンテンツ C. タブ付きのパネル Widget D. タブ付きのパネル

タブ付きのパネル Widget の HTML コードは、すべてのパネルが含まれる外側の div タグ、タブのリスト、コンテンツパネルが含まれる div タグ、および各コンテンツパネルの div タグで構成されます。タブ付きのパネル Widget の HTML コードには、ドキュメントのヘッド内とタブ付きのパネル Widget の HTML コードの後の script タグも含まれます。

ドキュメントのヘッド内の script タグは、タブ付きのパネル Widget に関連するすべての JavaScript 関数を定義します。タブ付きのパネル Widget のコードの後の script タグは、タブ付きのパネルをインタラクティブにする JavaScript オブジェクトを作成します。タブ付きのパネル Widget の HTML コードを次に示します。

```
<head>
. . .
  <!--Link the CSS style sheet that styles the tabbed panel-->
  <link href="SpryAssets/SpryTabbedPanels.css" rel="stylesheet" type="text/css" />
  <!--Link the Spry TabbedPanels JavaScript library-->
  <script src="SpryAssets/SpryTabbedPanels.js" type="text/javascript"></script>
</head>
<body>
  <!--Create the Tabbed Panel widget and assign classes to each element-->
  <div class="TabbedPanels" id="TabbedPanels1">
    <ul class="TabbedPanelsTabGroup">
      <li class="TabbedPanelsTab">Tab 1</li>
      <li class="TabbedPanelsTab">Tab 2</li>
      <li class="TabbedPanelsTab">Tab 3</li>
      <li class="TabbedPanelsTab">Tab 4</li>
    </ul>
    <div class="TabbedPanelsContentGroup">
      <div class="TabbedPanelsContent">Tab 1 Content</div>
      <div class="TabbedPanelsContent">Tab 2 Content</div>
      <div class="TabbedPanelsContent">Tab 3 Content</div>
      <div class="TabbedPanelsContent">Tab 4 Content</div>
    </div>
  </div>
  <!--Initialize the Tabbed Panel widget object-->
  <script type="text/javascript">
    var TabbedPanels1 = new Spry.Widget.TabbedPanels("TabbedPanels1");
  </script>
</body>
```

このコードでは、new JavaScript 演算子によって、タブ付きのパネル Widget オブジェクトが初期化され、div コンテンツが TabbedPanels1 の ID を使用して静的 HTML コードからインタラクティブなページエレメントに変換されます。Spry.Widget.TabbedPanels メソッドはタブ付きのパネルオブジェクトを作成する Spry フレームワークのコンストラクタです。このオブジェクトを初期化するために必要な情報は、ドキュメントのヘッド内でリンクした SpryTabbedPanels.js JavaScript ライブラリに格納されています。

タブ付きのパネル Widget の各エレメントには、CSS クラスが格納されています。これらのクラスはタブ付きのパネル Widget のスタイルを制御し、対応する "SpryTabbedPanels.css" ファイルに存在します。

タブ付きのパネル Widget の特定の部分の外観を変更するには、HTML コードでその部分に割り当てられたクラス名に対応する CSS ルールを編集します。たとえば、タブ付きのパネルのタブの背景色を変更するには、"SpryTabbedPanels.css" ファイルの TabbedPanelsTab ルールを編集します。"SpryTabbedPanels.css" ファイルの CSS コードを変更すると、そのファイルにリンクされているすべてのタブ付きのパネルが影響を受けます。

HTML コードに示されているクラスに加えて、タブ付きのパネル Widget には、Widget に関連付けられた特定のデフォルトのビヘイビアもあります。このビヘイビアは Spry フレームワークの組み込み部分で、"SpryTabbedPanels.js" JavaScript ライブラリファイルに格納されています。タブ付きのパネルライブラリには、マウスオーバー、パネルを開くためのタブのクリック、パネルのフォーカス、およびキーボードによる操作に関連するビヘイビアが格納されています。

タブ付きのパネルの外観を変更してこのビヘイビアに関連付けるには、"SpryTabbedPanels.css" ファイルの適切なクラスを編集します。特定のビヘイビアを削除する場合は、そのビヘイビアに対応する CSS ルールを削除できます。

**注意：**タブ付きのパネルの外観を変更して特定のビヘイビアに関連付けることはできますが、組み込みビヘイビア自体は変更できません。たとえば、"SpryTabbedPanels.css" ファイルの TabbedPanelsContentVisible クラスにプロパティが設定されていない場合でも、現在開いているパネルに TabbedPanelsContentVisible クラスが適用されます。

### タブ付きのパネル Widget 構造に使用されるさまざまなタグ

上の例では、div タグおよびリストアイテムによって Widget のネストされたタグ構造が作成されます。

```
Container <div>
  Tabs <ul>
    Tab <li>
      Content container <div>
        Content <div>
```

タブ付きのパネル Widget が正常に機能するためには、この 3 レベルの 5 つのコンテナ構造が不可欠です。ただし、この構造は、使用する実際のタグより重要です。Spry で構造 (div タグとは限りません) が読み取られ、それによって Widget が作成されます。3 レベルの 5 つのコンテナ構造が適切に配置されていれば、ブロックレベルのエレメントを使用して Widget を作成できます。

```
Container <div>
  Tabs <div>
    Tab <h3>
      Content container <div>
        Content <p>
```

この例の div タグは柔軟で、他のブロックレベルのエレメントを格納できます。ただし、p タグまたはその他のインラインタグには他のブロックレベルのエレメントを格納できないため、このタグを、タブ付きのパネルのコンテナまたはパネルタグとして使用することはできません。

### タブ付きのパネル Widget の CSS コード

"SpryTabbedPanels.css" ファイルには、タブ付きのパネル Widget のスタイルを設定するルールが含まれています。このルールを編集して、タブ付きのパネルの外観と印象に関するスタイル設定を実行できます。CSS ファイルのルールの名前は、タブ付きのパネル Widget の HTML コードで指定されたクラスの名前に直接対応します。

**注意：**"SpryTabbedPanels.css" ファイルおよび HTML コードのスタイル関連のクラス名を独自のクラス名に置き換えることができます。CSS コードは Widget の機能には不要であるため、置き換えても Widget の機能には影響しません。

スタイルシートの末尾にあるビヘイビアクラスのデフォルト機能は、"SpryTabbedPanels.js" JavaScript ライブラリファイルで定義されます。デフォルト機能を変更するには、スタイルシートのビヘイビアルールにプロパティおよび値を追加します。

"SpryTabbedPanels.css" ファイルの CSS コードを次に示します。ファイルの前半には、水平タブ付きのパネルに関するスタイル設定ルールが格納されています。ファイルの後半には、垂直タブ付きのパネルに関するスタイル設定ルールが格納されています。

```
/* Horizontal Tabbed Panels */
.TabbedPanels {
    margin: 0px;
    padding: 0px;
    clear: both;
    width: 100%; /* IE Hack to force proper layout when preceded by a paragraph. (hasLayout Bug) */
}
.TabbedPanelsTabGroup {
    margin: 0px;
    padding: 0px;
}
.TabbedPanelsTab {
    position: relative;
    top: 1px;
    float: left;
    padding: 4px 10px;
    margin: 0px 1px 0px 0px;
    font: bold 0.7em sans-serif;
    background-color: #DDD;
    list-style: none;
    border-left: solid 1px #CCC;
    border-bottom: solid 1px #999;
    border-top: solid 1px #999;
    border-right: solid 1px #999;
    -moz-user-select: none;
    -khtml-user-select: none;
    cursor: pointer;
}
.TabbedPanelsTabHover {
    background-color: #CCC;
}
.TabbedPanelsTabSelected {
    background-color: #EEE;
    border-bottom: 1px solid #EEE;
}
.TabbedPanelsTab a {
    color: black;
    text-decoration: none;
}
.TabbedPanelsContentGroup {
    clear: both;
    border-left: solid 1px #CCC;
    border-bottom: solid 1px #CCC;
    border-top: solid 1px #999;
    border-right: solid 1px #999;
    background-color: #EEE;
}
.TabbedPanelsContent {
    padding: 4px;
}
.TabbedPanelsContentVisible {
}
/* Vertical Tabbed Panels */
.VTabbedPanels .TabbedPanelsTabGroup {
    float: left;
    width: 10em;
    height: 20em;
    background-color: #EEE;
}
```

```

    position: relative;
    border-top: solid 1px #999;
    border-right: solid 1px #999;
    border-left: solid 1px #CCC;
    border-bottom: solid 1px #CCC;
}
.VTabbedPanels .TabbedPanelsTab {
    float: none;
    margin: 0px;
    border-top: none;
    border-left: none;
    border-right: none;
}
.VTabbedPanels .TabbedPanelsTabSelected {
    background-color: #EEE;
    border-bottom: solid 1px #999;
}
.VTabbedPanels .TabbedPanelsContentGroup {
    clear: none;
    float: left;
    padding: 0px;
    width: 30em;
    height: 20em;
}
}

```

"SpryTabbedPanels.css" ファイルには、特定のルールに対するコードおよび目的を説明するさまざまなコメントが含まれます。詳細については、ファイル内のコメントを参照してください。

## タブ付きのパネル Widget の挿入

**1** "SpryTabbedPanels.js" ファイルを検索して、Web サイトに追加します。"SpryTabbedPanels.js" ファイルは、Adobe Labs からダウンロードした Spry ディレクトリにある `widgets/tabbedpanels` ディレクトリ内で検索できます。詳細については、3 ページの「ファイルの準備」を参照してください。

たとえば、Web サイトのルートフォルダ内に "**SpryAssets**" という名前のフォルダを作成し、このフォルダに "SpryTabbedPanels.js" ファイルを移動します。"SpryTabbedPanels.js" ファイルには、タブ付きのパネル Widget をインタラクティブにするために必要な情報がすべて含まれています。

**2** "SpryTabbedPanels.css" ファイルを検索して、Web サイトに追加します。このファイルの追加先として、メインディレクトリである `SpryAssets` ディレクトリ、または CSS ディレクトリがある場合はそのディレクトリを選択できます。

**3** タブ付きのパネル Widget の追加先となる Web ページを開き、このページのヘッドタグに次の script タグを挿入してこのページに "SpryTabbedPanels.js" ファイルをリンクします。

```
<script src="SpryAssets/SpryTabbedPanels.js" type="text/javascript"></script>
```

"SpryTabbedPanels.js" ファイルへのファイルパスが正しいことを確認します。このパスは、Web サイト内のどこにファイルを配置したかによって異なります。

**4** ヘッドタグに次の link タグを挿入して、Web ページに "SpryTabbedPanels.css" ファイルをリンクします。

```
<link href="SpryAssets/SpryTabbedPanels.css" rel="stylesheet" type="text/css" />
```

"SpryTabbedPanels.css" ファイルへのファイルパスが正しいことを確認します。このパスは、Web サイト内のどこにファイルを配置したかによって異なります。

**5** タブ付きのパネル Widget を Web ページに追加するには、ページ上のタブ付きのパネルの表示位置に次の div タグを挿入します。

```
<div id="TabbedPanels1" class="TabbedPanels">
</div>
```

**6** タブ付きのパネルにタブを追加するには、次のように、`ul class="TabbedPanelsTabGroup"` タグを `div id="TabbedPanels1"...` タグ内に挿入し、追加するタブごとに `li class="TabbedPanelsTab"` タグを挿入します。

```
<div class="TabbedPanels" id="TabbedPanels1">
  <ul class="TabbedPanelsTabGroup">
    <li class="TabbedPanelsTab">Tab 1</li>
    <li class="TabbedPanelsTab">Tab 2</li>
    <li class="TabbedPanelsTab">Tab 3</li>
    <li class="TabbedPanelsTab">Tab 4</li>
  </ul>
</div>
```

上のコードでは、Widget に 4 つのタブが追加されます。追加できるタブ数は無制限です。

**7** 各タブにコンテンツ領域やパネルを追加するには、次のように、div class="TabbedPanelsContentGroup" コンテナタグを ul タグの後に挿入し、コンテンツパネルごとに div class="TabbedPanelsContent" タグを挿入します。

```
<div class="TabbedPanels" id="TabbedPanels1">
  <ul class="TabbedPanelsTabGroup">
    <li class="TabbedPanelsTab">Tab 1</li>
    <li class="TabbedPanelsTab">Tab 2</li>
    <li class="TabbedPanelsTab">Tab 3</li>
    <li class="TabbedPanelsTab">Tab 4</li>
  </ul>
  <div class="TabbedPanelsContentGroup">
    <div class="TabbedPanelsContent">Tab 1 Content</div>
    <div class="TabbedPanelsContent">Tab 2 Content</div>
    <div class="TabbedPanelsContent">Tab 3 Content</div>
    <div class="TabbedPanelsContent">Tab 4 Content</div>
  </div>
</div>
```

TabbedPanelsContent の開始タグと終了タグの間にコンテンツを挿入します。たとえば、前の例では Tab 1 Content です。コンテンツには、HTML フォームエレメントなどの有効な HTML コードを指定できます。

**8** Spry タブ付きのパネルオブジェクトを初期化するには、次の script ブロックをタブ付きのパネル Widget に対応する HTML コードの後に挿入します。

```
<div id="TabbedPanels1" class="TabbedPanels">
  . . .
</div>
<script type="text/javascript">
  var TabbedPanels1 = new Spry.Widget.TabbedPanels("TabbedPanels1");
</script>
```

new JavaScript 演算子によって、タブ付きのパネル Widget オブジェクトが初期化され、div コンテンツが TabbedPanels1 の ID を使用して静的 HTML コードからインタラクティブなタブ付きのパネルオブジェクトに変換されます。

Spry.Widget.TabbedPanels メソッドは、タブ付きのパネルオブジェクトを作成する Spry フレームワークのコンストラクタです。このオブジェクトを初期化するために必要な情報は、この手順の開始時にリンクした SpryTabbedPanels.js JavaScript ライブラリに含まれています。

タブ付きのパネルの div タグの ID が、Spry.Widgets.TabbedPanels メソッドで指定した ID パラメータと一致することを確認します。JavaScript の呼び出しが Widget に対応する HTML コードの後に行われることを確認します。

**9** ページを保存します。

コード全体を次に示します。

```
<head>
. . .
  <link href="SpryAssets/SpryTabbedPanels.css" rel="stylesheet" type="text/css" />
  <script src="SpryAssets/SpryTabbedPanels.js" type="text/javascript"></script>
</head>
<body>
  <div class="TabbedPanels" id="TabbedPanels1">
    <ul class="TabbedPanelsTabGroup">
      <li class="TabbedPanelsTab">Tab 1</li>
      <li class="TabbedPanelsTab">Tab 2</li>
      <li class="TabbedPanelsTab">Tab 3</li>
      <li class="TabbedPanelsTab">Tab 4</li>
    </ul>
    <div class="TabbedPanelsContentGroup">
      <div class="TabbedPanelsContent">Tab 1 Content</div>
      <div class="TabbedPanelsContent">Tab 2 Content</div>
      <div class="TabbedPanelsContent">Tab 3 Content</div>
      <div class="TabbedPanelsContent">Tab 4 Content</div>
    </div>
  </div>
  <script type="text/javascript">
    var TabbedPanels1 = new Spry.Widget.TabbedPanels("TabbedPanels1");
  </script>
</body></body>
```

## タブ付きのパネル Widget へのパネルの追加

❖ `li class="TabbedPanelsTab"` タグを、タブの `ul` リストに追加します。また、`div class="TabbedPanelsContent"` タグを、パネルコンテンツのコンテナである `div` タグに追加します。コードを追加する際、必ず終了タグ `/li` および `/div` も追加してください。以下に例を挙げます。

```
<div class="TabbedPanels" id="TabbedPanels1">
  <ul class="TabbedPanelsTabGroup">
    <li class="TabbedPanelsTab">Tab 1</li>
    <li class="TabbedPanelsTab">Tab 2</li>
  </ul>
  <div class="TabbedPanelsContentGroup">
    <div class="TabbedPanelsContent">Tab 1 Content</div>
    <div class="TabbedPanelsContent">Tab 2 Content</div>
  </div>
</div>
```

追加できるパネル数は無制限です。TabbedPanelsTab*i* 項目の数と TabbedPanelsContent*div* タグの数の比率は、常に 1:1 にする必要があります。

## タブ付きのパネル Widget からのパネルの削除

❖ 目的の `li class="TabbedPanelsTab"` タグと、対応する `<div class="TabbedPanelsContent">` をコードから削除します。コードを削除する際、必ず終了タグ `</li>` および `</div>` も削除してください。

## キーボードによる操作の有効化

キーボードによる操作で Widget にアクセスできるようにすることは、すべての Widget にとって重要です。キーボードによる操作では、ユーザーが矢印キーやカスタムキーを使用して Widget を制御できます。

キーボードによる操作の基礎は `tabIndex` 属性です。この属性は、タブを使用してドキュメント内を移動する方法をブラウザに指示します。

❖ タブ付きのパネルでキーボードによる操作を有効にするには、次のように、`TabIndex` 値を各 `li` タグに追加します。

```
<ul class="TabbedPanelsTabGroup">
  <li class="TabbedPanelTab" tabIndex="0">Tab</li>
  <li class="TabbedPanelTab" tabIndex="0">Tab</li>
</ul>
```

tabIndex 属性の値がゼロ (0) の場合、ブラウザで順序が決定されます。正の整数値の場合、その順序値が使用されます。

**注意:** div タグでの tabIndex の使用は XHTML 1.0 に準拠していません。

## タブ付きのパネルの方向の変更

タブ付きのパネルはデフォルトで水平に表示されますが、垂直タブ付きのパネルも簡単に作成できます。

❖ 水平タブ付きのパネル Widget を垂直タブ付きのパネル Widget に変更するには、次のように、メインコンテナの div タグのクラスを、TabbedPanels から VTabbedPanels に変更します。

```
<div class="VTabbedPanels" id="TabbedPanels1">
  <ul class="TabbedPanelsTabGroup">
    <li class="TabbedPanelsTab">Tab 1</li>
    <li class="TabbedPanelsTab">Tab 2</li>
    . . .
  </ul>
</div>
```

## 最初に開くパネルの設定

タブ付きのパネル Widget を含むページがブラウザにロードされるときに特定のパネルが開くように設定できます。

❖ 次のように、コンストラクタで defaultTab オプションを設定します。

```
<script type="text/javascript">
  var TabbedPanels1 = new Spry.Widget.TabbedPanels("TabbedPanels1", { defaultTab: 2 });
</script>
```

**注意:** タブ付きのパネル Widget ではゼロからカウントを開始するシステムが使用されるため、値を 2 に設定すると 3 番目のタブ付きのパネルが開きます。

## プログラムによるパネルの表示

特定のパネルをプログラムで開くには、JavaScript 関数を使用します。たとえば、クリックすると特定のタブ付きのパネルが開かれるボタンをページに配置できます。

Spry ではゼロからカウントを開始するシステムが使用されています。そのため、値 0 は、左端の最初のタブ付きのパネルを表します。タブ付きのパネルに ID がある場合、ID を使用してパネルを参照することもできます。

❖ 次の関数を使用して特定のタブ付きのパネルを開きます。

```
<button onclick="TabbedPanels1.showPanel(0)" >open first panel</button>
<button onclick="TabbedPanels1.showPanel('tabID')">open panel</button>
<script type="text/javascript">
  var TabbedPanels1 = new Spry.Widget.TabbedPanels("TabbedPanels1");
</script>
```

## タブ付きのパネル Widget のカスタマイズ

"SpryTabbedPanels.css" ファイルには、タブ付きのパネル Widget のデフォルトのスタイル設定が用意されています。ただし、該当する CSS ルールを変更することによって、Widget を簡単にカスタマイズできます。"SpryTabbedPanels.css" ファイルの CSS ルールでは、タブ付きのパネルの HTML コードの関連するエレメントと同じクラス名が使用されるため、タブ付きのパネル Widget のさまざまなセクションに対応する CSS ルールを簡単に把握できます。さらに、"SpryTabbedPanels.css" ファイルには、Widget に関連するビヘイビア (マウスオーバーやクリックのビヘイビアなど) のクラス名が含まれています。

"SpryTabbedPanels.css" ファイルは、カスタマイズを開始する前に Web サイトに配置しておく必要があります。詳細については、3 ページの「ファイルの準備」を参照してください。

**注意:** Internet Explorer バージョン 6 以前では、兄弟および子のコンテキストセレクタ (.TabbedPanels + .TabbedPanels や .TabbedPanels > .TabbedPanels など) はサポートされていません。

### タブ付きのパネル Widget のスタイル設定 (一般的な方法)

タブ付きのパネル Widget のコンテナ全体のプロパティを設定するか、Widget の各コンポーネントのプロパティを個別に設定します。

- 1 "SpryTabbedPanels.css" ファイルを開きます。
- 2 タブ付きのパネルの変更する部分の CSS ルールを見つけます。たとえば、タブ付きのパネルのタブの背景色を変更するには、"SpryTabbedPanels.css" ファイルの TabbedPanelsTab ルールを編集します。
- 3 CSS ルールを変更してファイルを保存します。

"SpryTabbedPanels.css" ファイルおよび HTML コードのスタイル関連のクラス名を独自のクラス名に置き換えることができます。置き換えても Widget の機能には影響しません。

"SpryTabbedPanels.css" ファイルには、特定のルールに対するコードおよび目的を説明するさまざまなコメントが含まれます。詳細については、ファイル内のコメントを参照してください。

### タブ付きのパネル Widget のテキストのスタイル設定

タブ付きのパネル Widget のコンテナ全体のプロパティを設定するか、Widget の各コンポーネントのプロパティを個別に設定します。

❖ 次の表を使用して適切な CSS ルールを見つけ、独自のテキストスタイルプロパティおよび値を追加します。

| 変更するテキスト        | 関連する CSS ルール  | 追加するプロパティおよび値の例                |
|-----------------|---|--------------------------------|
| Widget 全体のテキスト  | .TabbedPanels   | font: Arial; font-size:medium; |
| パネルタブのみのテキスト    | .TabbedPanelsTabGroup または<br>.TabbedPanelsTab         | font: Arial; font-size:medium; |
| コンテンツパネルのみのテキスト | .TabbedPanelsContentGroup または<br>.TabbedPanelsContent | font: Arial; font-size:medium; |

### タブ付きのパネル Widget の背景色の変更

❖ 次の表を使用して適切な CSS ルールを見つけ、背景色プロパティおよび値を追加または変更します。

| 変更する色                    | 関連する CSS ルール  | 追加または変更するプロパティおよび値の例                 |
|--------------------------|---|--------------------------------------|
| パネルタブの背景色                | .TabbedPanelsTabGroup または<br>.TabbedPanelsTab         | background-color: #DDD; (これが初期設定値です) |
| コンテンツパネルの背景色             | .TabbedPanelsContentGroup または<br>.TabbedPanelsContent | background-color: #EEE; (これが初期設定値です) |
| 選択されたタブの背景色              | .TabbedPanelsTabSelected                              | background-color: #EEE; (これが初期設定値です) |
| マウスポインタが上にあるときのパネルタブの背景色 | .TabbedPanelsTabHover                                 | background-color: #CCC; (これが初期設定値です) |

### タブ付きのパネルの幅の固定

初期設定では、タブ付きのパネル Widget は使用可能なスペースいっぱいに拡張されます。タブ付きのパネル Widget の幅を制限するには、コンテナの幅プロパティを設定します。

- 1 "SpryTabbedPanels.css" ファイルで TabbedPanels CSS ルールを見つけます。このルールは、タブ付きのパネル Widget のメインコンテナエレメントに関するプロパティを定義します。
- 2 このルールに、width: 300px; のように幅プロパティと値を追加します。

### タブ付きのパネルの高さの変更

デフォルトでは、タブ付きのパネル Widget の高さはコンテンツに応じて拡張されます。パネルを特定の高さに設定するには、TabbedPanelsContent ルールに高さプロパティを追加します。

- 1 "SpryTabbedPanels.css" ファイルで TabbedPanelsContent CSS ルールを見つけます。
- 2 このルールに、height: 300px; のように高さプロパティと値を追加します。

### タブ付きのパネルのビヘイビアの変更

タブ付きのパネル Widget には、いくつかの定義済みビヘイビアがあります。これらのビヘイビアは、特定のイベントが発生したときの CSS クラスの変更で構成されます。たとえば、マウスポインタがパネルタブの上に置かれると、TabbedPanelsTabHover クラスが Widget に適用されます。このビヘイビアは、リンクの a:hover に似ています。ビヘイビアは、デフォルトでは空白です。変更するには、ルールにプロパティおよび値を追加します。

- 1 "SpryTabbedPanels.css" ファイルを開き、変更するタブ付きのパネルビヘイビアの CSS ルールを見つけます。タブ付きのパネル Widget には、ビヘイビアの 4 つの組み込みルールがあります。

| ビヘイビア                       | 説明                             |
|-----------------------------|--------------------------------|
| .Tabbed PanelsTabHover      | マウスポインタをパネルタブの上に置くとアクティブになります。 |
| .Tabbed PanelsTabFocused    | タブにキーボードフォーカスを置くとアクティブになります    |
| .Tabbed PanelsTabSelected   | 現在選択されているタブでアクティブになります         |
| .TabbedPanelsContentVisible | 現在選択されているタブのコンテンツ領域でアクティブになります |

例については、Adobe Labs からダウンロードした Spry ディレクトリの samples ディレクトリにあるタブ付きのパネルのサンプルファイルを参照してください。詳細については、3 ページの「ファイルの準備」を参照してください。

- 2 有効にするビヘイビアの CSS ルールを追加します。

**注意：**ビヘイビアは Spry フレームワークの一部としてハードコーディングされているため、"SpryTabbedPanels.css" ファイルのビヘイビア関連のクラス名を独自のクラス名に置き換えることはできません。デフォルトのクラスを独自のクラス名で上書きするには、新しい値をパラメータとしてタブ付きのパネルコンストラクタに送ります。

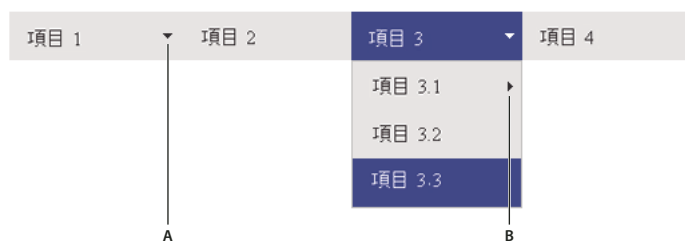
```
<script type="text/javascript">
    var TabbedPanels1 = new Spry.Widget.TabbedPanels("TabbedPanels1", { tabHoverClass: "hover",
panelVisibleClass: "open", tabSelectedClass: "selected", tabFocusedClass: "focused" });
</script>
```

## メニューバー Widget の操作

### メニューバー Widget の概要と構造

メニューバー Widget は、ボタンの 1 つにマウスポインタを置いたときにサブメニューが表示される、ナビゲーションメニューボタンのセットです。メニューバーを使用すると、わずかなスペースに多くのナビゲーション情報を表示できるだけでなく、ユーザーに長い距離をブラウズさせることなくサイト上の情報の全体像を把握させることができます。

次に水平メニューバー Widget の例を示します。この例では 3 番目のメニュー項目が展開されています。



メニューバー Widget (<ul>、<li>、および <a> タグで構成されます)

A. メニュー項目のサブメニュー B. サブメニュー項目のサブメニュー

メニューバー Widget の HTML コードは、外側の ul タグと、最上位レベルのメニュー項目ごとにこのタグに含まれる li タグで構成されます。一方、最上位レベルのメニュー項目 (li タグ) には、各項目のサブメニューを定義する ul タグおよび li タグが含まれます。同様に、サブメニューにサブメニューを含めることもできます。最上位レベルのメニューでもサブメニューでも、追加できるサブメニュー項目数は無制限です。

**注意：**1 つのメニューバー内のさまざまなレベルでサイトのすべてのページを表示することは、お勧めしません。よくあることですが、ユーザーがブラウザで JavaScript の機能をオフにしていた場合、表示されるのはメニューバーの最上位レベルのメニュー項目だけです。

メニューバー Widget の HTML コードには、ドキュメントのヘッド内とメニューバー Widget の HTML コードの後に script タグも含まれます。これらのタグでは、メニューバーをインタラクティブにする JavaScript オブジェクトを作成します。メニューバー Widget を垂直にするか水平にするかは、メニューバーのメインコンテンツの ul タグで定義します。水平メニューバー Widget の HTML コードを次に示します。

```
<head>
...
<!--Link the Spry Menu Bar JavaScript library-->
<script src="SpryAssets/SpryMenuBar.js" type="text/javascript"></script>
<!--Link the CSS style sheet that styles the menu bar. You can select between horizontal and vertical-->
<link href="SpryAssets/SpryMenuBarHorizontal.css" rel="stylesheet" type="text/css" />
</head>
<body>
<!--Create a Menu bar widget and assign classes to each element-->
<ul id="menubar1" class="MenuBarHorizontal">
  <li><a class="MenuBarItemSubmenu" href="#">Item 1</a>
    <ul>
      <li><a href="#">Item 1.1</a></li>
      <li><a href="#">Item 1.2</a></li>
      <li><a href="#">Item 1.3</a></li>
    </ul>
  </li>
  <li><a href="#">Item 2</a></li>
  <li><a class="MenuBarItemSubmenu" href="#">Item 3</a>
    <ul>
      <li><a class="MenuBarItemSubmenu" href="#">Item 3.1</a>
        <ul>
          <li><a href="#">Item 3.1.1</a></li>
          <li><a href="#">Item 3.1.2</a></li>
        </ul>
      </li>
      <li><a href="#">Item 3.2</a></li>
      <li><a href="#">Item 3.3</a></li>
    </ul>
  </li>
  <li><a href="#">Item 4</a></li>
</ul>
<!--Initialize the Menu Bar widget object-->
<script type="text/javascript">
  var menubar1 = new Spry.Widget.MenuBar("menubar1", {imgDown:"SpryAssets/SpryMenuBarDownHover.gif",
imgRight:"SpryAssets/SpryMenuBarRightHover.gif"});
</script>
</body>
```

このコードでは、new JavaScript 演算子によって、メニューバーオブジェクトが初期化され、ul コンテンツが menuBar1 の ID を使用して静的 HTML コードからインタラクティブなページエレメントに変換されます。Spry.Widget.MenuBar メソッドはメニューバーオブジェクトを作成する Spry フレームワークのコンストラクタです。このオブジェクトを初期化するために必要な情報は、ドキュメントのヘッド内でリンクした MenuBar.js JavaScript ライブラリに格納されています。

Widget を作成する a タグの多くには、CSS クラスが格納されています。このクラスはメニューバー Widget のスタイルを制御し、対応する CSS ファイル、つまり選択に応じて "SpryMenuBarHorizontal.css" か "SpryMenuBarVertical.css" のいずれかのファイルに存在します。

メニューバー Widget の特定の部分の外観を変更するには、HTML コードでその部分に割り当てられたクラス名に対応する CSS ルールを編集します。たとえば、メニューバーの最上位レベルのメニュー項目の背景色を変更するには、"SpryMenuBarHorizontal.css" ファイルの対応する CSS ルールを編集します。"SpryManuBarHorizontal.css" ファイルの CSS コードを変更すると、そのファイルにリンクされているすべてのメニューバーが影響を受けます。

HTML コードに表示されているクラスに加えて、メニューバー Widget には、Widget に関連付けられた特定のデフォルトのビヘイビアもあります。このビヘイビアは Spry フレームワークの組み込み部分で、"SpryMenuBar.js" JavaScript ライブラリファイルに格納されています。このライブラリファイルには、マウスオーバー関連のビヘイビアが格納されています。

メニューバーの外観を変更してこのビヘイビアに関連付けるには、いずれかの CSS ファイルの適切なクラスを編集します。何らかの理由で特定のビヘイビアを削除する場合は、そのビヘイビアに対応する CSS ルールを削除できます。

## メニューバー Widget の CSS コード

"SpryMenuBarHorizontal.css" ファイルと "SpryMenuBarVertical.css" ファイルには、メニューバー Widget のスタイルを設定するルールが含まれます。このルールを編集して、メニューバーの外観と印象に関するスタイル設定を実行できます。CSS ファイル内のルールの名前は、メニューバー Widget の HTML コードで指定されたクラスの名前に直接対応します。

**注意:** "SpryMenuBarHorizontal.css" ファイル、"SpryMenuBarVertical.css" ファイルおよび HTML コードのスタイル関連のクラス名を独自のクラス名に置き換えることができます。CSS コードは Widget の機能には不要であるため、置き換えても Widget の機能には影響しません。

スタイルシートの末尾にあるビヘイビアクラスのデフォルト機能は、"SpryMenuBar.js" JavaScript ライブラリファイルで定義されます。デフォルト機能を変更するには、スタイルシートのビヘイビアルールにプロパティおよび値を追加します。

"SpryMenuBarHorizontal.css" ファイルの CSS コードを次に示します。

```
/*Menu Bar styling classes*/
ul.MenuBarHorizontal{
    margin: 0;
    padding: 0;
    list-style-type: none;
    font-size: 100%;
    cursor: default;
    width: auto;
}
ul.MenuBarActive{
    z-index: 1000;
}
ul.MenuBarHorizontal li{
    margin: 0;
    padding: 0;
    list-style-type: none;
    font-size: 100%;
    position: relative;
    text-align: left;
    cursor: pointer;
    width: 8em;
    float: left;
}
ul.MenuBarHorizontal ul{
```

```
margin: 0;
padding: 0;
list-style-type: none;
font-size: 100%;
z-index: 1020;
cursor: default;
width: 8.2em;
position: absolute;
left: -1000em;
}
ul.MenuBarHorizontal ul.MenuBarSubmenuVisible{
  left: auto;
}
ul.MenuBarHorizontal ul li{
  width: 8.2em;
}
ul.MenuBarHorizontal ul ul{
  position: absolute;
  margin: -5% 0 0 95%;
}
ul.MenuBarHorizontal ul.MenuBarSubmenuVisible ul.MenuBarSubmenuVisible{
  left: auto;
  top: 0;
}
ul.MenuBarHorizontal ul{
  border: 1px solid #CCC;
}
ul.MenuBarHorizontal a{
  display: block;
  cursor: pointer;
  background-color: #EEE;
  padding: 0.5em 0.75em;
  color: #333;
  text-decoration: none;
}
ul.MenuBarHorizontal a:hover, ul.MenuBarHorizontal a:focus{
  background-color: #33C;
  color: #FFF;
}
ul.MenuBarHorizontal a.MenuBarItemHover, ul.MenuBarHorizontal a.MenuBarItemSubmenuHover,
ul.MenuBarHorizontal a.MenuBarSubmenuVisible{
  background-color: #33C;
  color: #FFF;
}
ul.MenuBarHorizontal a.MenuBarItemSubmenu{
  background-image: url(SpryMenuBarDown.gif);
  background-repeat: no-repeat;
  background-position: 95% 50%;
}
ul.MenuBarHorizontal ul a.MenuBarItemSubmenu{
  background-image: url(SpryMenuBarRight.gif);
  background-repeat: no-repeat;
  background-position: 95% 50%;
}
ul.MenuBarHorizontal a.MenuBarItemSubmenuHover{
```

```

        background-image: url(SpryMenuBarDownHover.gif);
        background-repeat: no-repeat;
        background-position: 95% 50%;
    }
    ul.MenuBarHorizontal ul a.MenuBarItemSubmenuHover{
        background-image: url(SpryMenuBarRightHover.gif);
        background-repeat: no-repeat;
        background-position: 95% 50%;
    }
}
ul.MenuBarHorizontal iframe{
    position: absolute;
    z-index: 1010;
}
}
@media screen, projection{
    ul.MenuBarHorizontal li.MenuBarItemIE{
        display: inline;
        float: left;
        background: #FFF;
    }
}
}

```

"SpryMenuBar.css" ファイルには、特定のルールに対するコードおよび目的を説明するさまざまなコメントが含まれます。詳細については、ファイル内のコメントを参照してください。

## メニューバー Widget の挿入

**1** "SpryMenuBar.js" ファイルを検索して、Web サイトに追加します。"SpryMenuBar.js" ファイルは、Adobe Labs からダウンロードした Spry ディレクトリにある widgets/menubar ディレクトリ内で検索できます。詳細については、3 ページの「ファイルの準備」を参照してください。

たとえば、Web サイトのルートフォルダ内に **SpryAssets** という名前のフォルダを作成し、このフォルダに "SpryMenuBar.js" ファイルを移動します。"SpryMenuBar.js" ファイルには、メニューバー Widget をインタラクティブにするために必要な情報がすべて含まれています。

**2** 作成するメニューバーの種類に応じて、"SpryMenuBarHorizontal.css" ファイルまたは "SpryMenuBarVertical.css" ファイルのいずれかを検索して、Web サイトに追加します。このファイルの追加先として、メインディレクトリである SpryAssets ディレクトリ、または CSS ディレクトリがある場合はそのディレクトリを選択できます。

**3** メニューバー Widget の追加先となる Web ページを開き、このページのヘッドタグに次の script タグを挿入してこのページに "SpryMenuBar.js" ファイルをリンクします。

```
<script src="SpryAssets/SpryMenuBar.js" type="text/javascript"></script>
```

"SpryMenuBar.js" ファイルへのファイルパスが正しいことを確認します。このパスは、Web サイト内のどこにファイルを配置したかによって異なります。

**4** ページのヘッドタグに次の link タグを挿入して Web ページに "SpryMenuBarHorizontal.css" ファイルまたは "SpryMenuBarVertical.css" ファイルをリンクします。

```
<link href="SpryAssets/SpryMenuBarHorizontal.css" rel="stylesheet" type="text/css" />
```

"SpryMenuBarHorizontal.css" または "SpryMenuBarVertical.css" へのファイルパスが正しいことを確認します。このパスは、Web サイト内のどこにファイルを配置したかによって異なります。

**5** 次のように、ul タグをコンテナタグとして挿入し、メニューバーの最上位レベルのメニュー項目ごとに li タグとサンプルテキストを挿入して Web ページにメニューバーの HTML コードを追加します。

```

<body>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
    <li>Item 4</li>
  </ul>
</body>

```

6 li タグのテキストを a タグで囲みます。

```
<body>
  <ul>
    <li><a href="#">Item 1</a></li>
    <li><a href="#">Item 2</a></li>
    <li><a href="#">Item 3</a></li>
    <li><a href="#">Item 4</a></li>
  </ul>
</body>
```

7 次のように、3 番目のメニュー項目 ( または他の任意のメニュー項目 ) に、a タグを含むリストをもう一つネストします。

```
<body>
  <ul>
    <li><a href="#">Item 1</a></li>
    <li><a href="#">Item 2</a></li>
    <li><a href="#">Item 3</a>
      <ul>
        <li><a href="#">Submenu Item 1</a></li>
        <li><a href="#">Submenu Item 2</a></li>
      </ul>
    </li>
    <li><a href="#">Item 4</a></li>
  </ul>
</body>
```

ネストされたこのリストは、3 番目のメニュー項目のサブメニューです。ネストされたリストが、最上位レベルのメニュー項目の a タグ内にないことを確認します。

8 メニューバーコンテナ ( ul タグ ) を識別する一意の ID を、次のように追加します。

```
<body>
  <ul id="menubar1">
    <li><a href="#">Item 1</a></li>
    <li><a href="#">Item 2</a></li>
    <li><a href="#">Item 3</a>
      <ul>
        <li><a href="#">Submenu Item 1</a></li>
        <li><a href="#">Submenu Item 2</a></li>
      </ul>
    </li>
    <li><a href="#">Item 4</a></li>
  </ul>
</body>
```

この ID は、後で Widget コンストラクタ内のコンテナの識別に使用します。

9 メニューバーにスタイル設定を追加するには、次のように、適切なクラスを追加します。

```
<body>
  <ul id="menubar1" class="MenuBarHorizontal">
    <li><a href="#">Item 1</a></li>
    <li><a href="#">Item 2</a></li>
    <li><a href="#" class="MenuBarItemSubmenu">Item 3</a>
      <ul>
        <li><a href="#">Submenu Item 1</a></li>
        <li><a href="#">Submenu Item 2</a></li>
      </ul>
    </li>
    <li><a href="#">Item 4</a></li>
  </ul>
</body>
```

水平メニューバーと垂直メニューバーのいずれを作成するかを指定します。最上位レベルのメニューバー項目にサブメニューがある場合は、MenuBarItemSubmenu クラスを a タグに割り当てます。このクラスでは下矢印イメージを表示して、サブメニューが存在することを示します。

**10 Spry メニューバーオブジェクトを初期化するには、次の script ブロックをメニューバー Widget に対応する HTML コードの後に挿入します。**

```
<ul id="menubar1" class="MenuBarHorizontal">
  . . .
</ul>
<script type="text/javascript">
  var menubar1 = new Spry.Widget.MenuBar("menubar1");
</script>
```

new JavaScript 演算子によって、メニューバー Widget オブジェクトが初期化され、ul コンテンツが menubar1 の ID を使用して静的 HTML コードからインタラクティブなメニューバーオブジェクトに変換されます。Spry.Widget.MenuBar メソッドはメニューバーオブジェクトを作成する Spry フレームワークのコンストラクタです。このオブジェクトを初期化するために必要な情報は、この手順の開始時にリンクした SpryMenuBar.js JavaScript ライブラリに含まれています。

メニューバーの ul コンテナタグの ID が、Spry.Widgets.MenuBar メソッドで指定した ID パラメータと一致することを確認します。JavaScript の呼び出しが Widget に対応する HTML コードの後に行われることを確認します。

**11 ページを保存します。**

コード全体を次に示します。

```
<head>
. . .
<script src="SpryAssets/SpryMenuBar.js" type="text/javascript"></script>
<link href="SpryAssets/SpryMenuBarHorizontal.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <ul id="menubar1" class="MenuBarHorizontal">
    <li><a href="#">Item 1</a></li>
    <li><a href="#">Item 2</a></li>
    <li><a href="#" class="MenuBarItemSubmenu">Item 3</a>
      <ul>
        <li><a href="#">Submenu Item 1</a></li>
        <li><a href="#">Submenu Item 2</a></li>
      </ul>
    </li>
    <li><a href="#">Item 4</a></li>
  </ul>
  <script type="text/javascript">
    var menubar1 = new Spry.Widget.MenuBar("menubar1");
  </script>
</body>
```

**注意：**Spry メニューバー Widget では、他のセクションの上に HTML コードのセクションを表示するため、DHTML レイヤーが使用されます。ページに Flash コンテンツが含まれる場合は、問題が生じる可能性があります。これは、Flash ムービーは常に他のすべての DHTML レイヤー上に表示されるため、Flash コンテンツがサブメニューの上に表示される可能性があるためです。この問題を解決するには、Flash コンテンツが wmode="transparent" を使用するようにパラメータを変更します。詳細については、[www.adobe.com/go/15523\\_jp](http://www.adobe.com/go/15523_jp) を参照してください。

## メニューおよびサブメニューの追加または削除

### メインメニュー項目の追加

❖ メインメニュー項目を追加するには、新しいリスト項目 (li タグ) をコンテナ ul タグに追加します。以下に例を挙げます。

```
<ul id="menubar1" class="MenuBarHorizontal">
  <li><a href="#">Item 1</a></li>
  <li><a href="#">Item 2</a></li>
  <li><a href="#" class="MenuBarItemSubmenu">Item 3</a>
    <ul>
      <li><a href="#">Submenu Item 1</a></li>
      <li><a href="#">Submenu Item 2</a></li>
    </ul>
  </li>
  <li><a href="#">Item 4</a></li>
  <li><a href="#">Item 5--NEW MENU ITEM</a></li>
</ul>
```

### サブメニュー項目の追加

❖ サブメニュー項目を追加するには、新しいリスト項目 (li タグ) をサブメニュー ul タグに追加します。以下に例を挙げます。

```
<ul id="menubar1" class="MenuBarHorizontal">
  <li><a href="#">Item 1</a></li>
  <li><a href="#">Item 2</a></li>
  <li><a href="#" class="MenuBarItemSubmenu">Item 3</a>
    <ul>
      <li><a href="#">Submenu Item 1</a></li>
      <li><a href="#">Submenu Item 2</a></li>
      <li><a href="#">Submenu Item 3--NEW SUBMENU ITEM</a></li>
    </ul>
  </li>
  <li><a href="#">Item 4</a></li>
</ul>
```

### メインメニュー項目またはサブメニュー項目の削除

❖ 削除するメニュー項目またはサブメニュー項目の li タグを削除します。

### メニュー項目のツールヒントの作成

❖ メニュー項目のツールヒントを作成するには、title 属性を該当するメニュー項目の a タグに追加します。以下に例を挙げます。

```
<ul id="menubar1" class="MenuBarHorizontal">
  <li><a href="#">Item 1</a></li>
  <li><a href="#">Item 2</a></li>
  <li><a href="#">Item 3</a></li>
  <li><a href="contacts.html" title="Contacts">Item 4</a></li>
</ul>
```

### イメージのプリロード

❖ サブメニューの下矢印および右矢印に使用するイメージをプリロードするには、imgDown オプションと imgRight オプションのいずれかまたは両方を Widget コンストラクタに追加します。以下に例を示します。

```
<script type="text/javascript">
  var menubar1 = new Spry.Widget.MenuBar("menubar1", {imgDown:"SpryAssets/SpryMenuBarDownHover.gif",
imgRight:"SpryAssets/SpryMenuBarRightHover.gif"});
</script>
```

適切なパスをオプションの値としてイメージに追加します。このパスは、イメージの保存場所によって異なります。

### メニューバー Widget の方向の変更

メニューバー Widget の方向を水平方向から垂直方向に、またはその逆に変更できます。これを行うには、メニューバーの HTML コードを変更し、Web サイトに適切な CSS ファイルがあることを確認します。

次の手順では、水平メニューバー Widget を垂直メニューバー Widget に変更しています。

**1** Web サイトに "SpryMenuBarVertical.css" ファイルがあることを確認します。たとえば、このファイルは、サイト内の "SpryAssets" フォルダに格納されている可能性があります。

**2** 次に示すように、ドキュメントのヘッド内にある "SpryMenuBarHorizontal.css" ファイルへのリンクを "SpryMenuBarVertical.css" ファイルへのリンクに置き換えます。

```
<link href="SpryAssets/SpryMenuBarVertical.css" rel="stylesheet" type="text/css" />
```

**3** 水平メニューバーの HTML コードで `MenuBarHorizontal` クラスを見つけ、それを `MenuBarVertical` に変更します。`MenuBarHorizontal` クラスは、メニューバーのコンテナ `ul` タグで定義されます (`<ul id="menubar1" class="MenuBarHorizontal">`)。

**4** メニューバーのコードの後に、メニューバーコンストラクタを見つけます。

```
var menubar1 = new Spry.Widget.MenuBar("menubar1", {imgDown:"SpryAssets/SpryMenuBarDownHover.gif",  
imgRight:"SpryAssets/SpryMenuBarRightHover.gif"});
```

**5** コンストラクタから `imgDown` プリロードオプション (および `canma`) を削除します。

```
var menubar1 = new Spry.Widget.MenuBar("menubar1", {imgRight:"SpryAssets/SpryMenuBarRightHover.gif"});
```

**注意:** 垂直メニューバーから水平メニューバーに変更する場合は、代わりに `imgDown` プリロードオプションと `canma` を追加します。

**6** (オプション) ページに他の水平メニューバー Widget が含まれない場合は、ドキュメントの `head` に埋め込まれている前の `MenuBarHorizontal.css` ファイルへのリンクを削除します。

**7** ページを保存します。

## メニューバー Widget のカスタマイズ

"SpryMenuBarHorizontal.css" ファイルと "SpryMenuBarVertical.css" ファイルには、メニューバー Widget のデフォルトのスタイル設定が用意されています。ただし、該当する CSS ルールを変更することによって Widget をカスタマイズできます。"SpryMenuBarHorizontal.css" ファイルと "SpryMenuBarVertical.css" ファイルの CSS ルールでは、メニューバーの HTML コードの関連するエレメントと同じクラス名が使用されるため、メニューバー Widget のさまざまなセクションに対応する CSS ルールを簡単に把握できます。さらに、"SpryMenuBarHorizontal.css" ファイルと "SpryMenuBarVertical.css" ファイルには、マウスオーバービヘイビアやクリックビヘイビアなど、Widget に関連するビヘイビアのクラス名が含まれています。

Widget の水平スタイルシートまたは垂直スタイルシートは、カスタマイズを開始する前に Web サイトに配置しておく必要があります。詳細については、3 ページの「ファイルの準備」を参照してください。

### メニューバー Widget のスタイル設定 (一般的な方法)

メニューバー Widget のスタイルを設定するには、メニューバー Widget コンテナ全体のプロパティを設定するか、Widget の各コンポーネントのプロパティを個別に設定します。

**1** "SpryMenuBarHorizontal.css" ファイルまたは "SpryMenuBarVertical.css" ファイルを開きます。

**2** メニューバーの変更する部分の CSS ルールを見つけます。たとえば、最上位レベルのメニュー項目の背景色を変更するには、CSS ファイルの `ulMenuBarHorizontal a` ルールまたは `ulMenuBarVertical a` ルールを編集します。

**3** CSS を変更してファイルを保存します。

CSS ファイルおよび HTML コードのスタイル関連のクラス名を独自のクラス名に置き換えることができます。置き換えても Widget の機能には影響しません。

"SpryMenuBarHorizontal.css" ファイルと "SpryMenuBarVertical.css" ファイルには、特定のルールに対するコードおよび目的を説明するさまざまなコメントが含まれます。詳細については、ファイル内のコメントを参照してください。

### メニュー項目のテキストスタイルの変更

a タグに関連付けられた CSS コードには、テキストスタイルに関する情報が含まれています。a タグには、さまざまなメニュー状態に関連するテキストスタイルクラス値が関連付けられています。

❖ メニュー項目のテキストスタイルを変更するには、次の表を使用して適切な CSS ルールを見つけ、デフォルト値を変更します。

| 変更するスタイル                  | 垂直または水平メニューバーに関する CSS ルール   | 関連するプロパティおよび初期設定値                   |
|---------------------------|---|-------------------------------------|
| 初期設定テキスト                  | ul.MenuBarVertical a,<br>ul.MenuBarHorizontal a   | color: #333; text-decoration: none; |
| マウスポインタが上にあるときのテキストの色     | ul.MenuBarVertical a:hover,<br>ul.MenuBarHorizontal a:hover                                     | color: #FFF;                        |
| フォーカスがあるときのテキストの色         | ul.MenuBarVertical a:focus,<br>ul.MenuBarHorizontal a:focus                                     | color: #FFF;                        |
| マウスポインタが上にあるときのメニューバー項目の色 | ul.MenuBarVertical a.MenuBarItemHover,<br>ul.MenuBarHorizontal a.MenuBarItemHover               | color: #FFF;                        |
| マウスポインタが上にあるときのサブメニュー項目の色 | ul.MenuBarVertical a.MenuBarItemSubmenuHover,<br>ul.MenuBarHorizontal a.MenuBarItemSubmenuHover | color: #FFF;                        |

### メニュー項目の背景色の変更

a タグに関連付けられた CSS ルールには、メニュー項目の背景色に関する情報が含まれています。a タグには、さまざまなメニュー状態に関連する背景色クラス値が関連付けられています。

❖ メニュー項目の背景色を変更するには、次の表を使用して適切な CSS ルールを見つけ、デフォルト値を変更します。

| 変更する色                     | 垂直または水平メニューバーに関する CSS ルール   | 関連するプロパティおよび初期設定値       |
|---------------------------|---|-------------------------|
| 初期設定の背景                   | ul.MenuBarVertical a,<br>ul.MenuBarHorizontal a   | background-color: #EEE; |
| マウスポインタが上にあるときの背景色        | ul.MenuBarVertical a:hover,<br>ul.MenuBarHorizontal a:hover                                     | background-color: #33C; |
| フォーカスがあるときの背景色            | ul.MenuBarVertical a:focus,<br>ul.MenuBarHorizontal a:focus                                     | background-color: #33C; |
| マウスポインタが上にあるときのメニューバー項目の色 | ul.MenuBarVertical a.MenuBarItemHover,<br>ul.MenuBarHorizontal a.MenuBarItemHover               | background-color: #33C; |
| マウスポインタが上にあるときのサブメニュー項目の色 | ul.MenuBarVertical a.MenuBarItemSubmenuHover,<br>ul.MenuBarHorizontal a.MenuBarItemSubmenuHover | background-color: #33C; |

### メニュー項目の寸法の変更

メニュー項目の寸法を変更するには、メニュー項目の li および ul タグの幅プロパティを変更します。

- 1 ul.MenuBarVertical li または ul.MenuBarHorizontal li ルールを見つけます。
- 2 幅プロパティを必要に応じて変更します。または、プロパティを auto に変更して固定幅を削除し、このルールにプロパティおよび値 white-space: nowrap; を追加します。
- 3 ul.MenuBarVertical ul または ul.MenuBarHorizontal ul ルールを見つけます。
- 4 幅プロパティを必要に応じて変更します。または、プロパティを auto に変更して固定幅を削除します。
- 5 ul.MenuBarVertical ul li または ul.MenuBarHorizontal ul li ルールを見つけます。

6 このルールにプロパティ `float: none;` および `background-color: transparent;` を追加します。

7 `width: 8.2em;` プロパティおよび値を削除します。

### サブメニューの位置の設定

Spry メニューバーのサブメニューの位置は、サブメニュー `ul` タグのマージンプロパティによって制御されます。

1 `ul.MenuBarVertical ul` または `ul.MenuBarHorizontal ul` ルールを見つけます。

2 `margin: -5% 0 95%;` の初期設定値を必要に応じて変更します。

## テキストフィールド検査 Widget の操作

### テキストフィールド検査 Widget の概要と構造

Spry テキストフィールド検査 Widget は、ユーザーがテキストを入力したときに有効か無効かを表示するテキストフィールドです。テキストフィールド検査 Widget は、たとえばユーザーが電子メールアドレスを入力するフォームに追加します。ユーザーが電子メールアドレスの「@」記号やピリオドを入力し忘れると、入力された情報が無効であることを示すメッセージが返されます。

次にテキストフィールド検査 Widget の例を示します。この例では有効状態を示しています。

A

B

C  無効な形式です。

D  値を指定する必要があります。

A. テキストフィールド Widget、ヒントのアクティブ化 B. テキストフィールド Widget、有効状態 C. テキストフィールド Widget、無効状態 D. テキストフィールド Widget、必須状態

テキストフィールド検査 Widget には、有効、無効、必須などのいくつかの状態が含まれます。これらの状態のプロパティは、使用する検査結果に応じて、対応する CSS ファイル (`SpryValidationTextField.css`) を編集することで変更できます。検査が実行されるタイミングは、さまざまに設定できます。たとえば、サイトビジターが Widget の外側をクリックしたとき、入力したとき、フォームを送信しようとしたときなどに設定できます。

**初期状態** ページがブラウザに読み込まれたとき、またはユーザーがフォームをリセットしたとき。

**フォーカス状態** ユーザーが Widget に挿入ポイントを置いたとき。

**有効状態** ユーザーが情報を正しく入力し、フォームを送信できる状態になったとき。

**無効状態** ユーザーが無効な形式のテキストを入力したとき。たとえば年のフィールドに「2006」ではなく「06」と入力したときの状態です。

**必須状態** ユーザーがテキストフィールドに必須テキストを入力しなかったとき。

**最小文字数状態** ユーザーが入力した文字数がテキストフィールドの最小文字数に満たなかったとき。

**最大文字数状態** ユーザーが入力した文字数がテキストフィールドの最大文字数を超過していたとき。

**最小値状態** ユーザーが入力した値がテキストフィールドの最小値に満たなかったとき。整数、実数、およびデータタイプの検査に適用されます。

**最大値状態** ユーザーが入力した値がテキストフィールドの最大値を超過していたとき。整数、実数、およびデータタイプの検査に適用されます。

ユーザーとのインタラクションの中で、テキストフィールド検査 Widget によってこれらのいずれかの状態が入力されると、実行時に Spry フレームワークロジックにより、この Widget の HTML コンテナに特定の CSS クラスが適用されます。たとえば、ユーザーがフォームを送信しようとしたときに必須テキストフィールドにテキストが入力されていない場合は、「値を指定する必要があります。」というエラーメッセージを表示するクラスが Widget に適用され、フォームの送信がブロックされます。エラーメッセージのスタイルと表示状態を制御するルールは、この Widget に対応する "SpryValidationTextField.css" ファイルに存在します。

テキストフィールド検査 Widget のデフォルト HTML コードは、通常はフォームの内部に配置され、コンテナ span タグで構成されます。このタグは、テキストフィールドの input タグを囲みます。テキストフィールド検査 Widget の HTML コードには、ドキュメントのヘッド内とこの Widget の HTML コードの後に script タグも含まれます。ドキュメントのヘッド内の script タグは、テキストフィールド Widget に関連するすべての JavaScript 関数を定義します。Widget コードの後の script タグは、テキストフィールドをインタラクティブにする JavaScript オブジェクトを作成します。

テキストフィールド検査 Widget の HTML コードを次に示します。

```
<head>
...
<!-- Link the Spry Validation Text Field JavaScript library -->
<script src="SpryAssets/SpryValidationTextField.js" type="text/javascript"></script>
<!-- Link the CSS style sheet that styles the widget -->
<link href="SpryAssets/SpryValidationTextField.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" name="form1" method="post" action="">
    <!-- Create the text field widget and assign a unique id-->
    <span id="sprytextfield1">
      <input type="text" name="mytextfield" id="mytextfield" />
      <!--Display an error message>
      <span class="textfieldRequiredMsg">A value is required.</span>
    </span>
  </form>
  <!-- Initialize the Validation Text Field widget object-->
  <script type="text/javascript">
var sprytextfield1 = new Spry.Widget.ValidationTextField("sprytextfield1");
</script>
</body>
```

このコードでは、new JavaScript 演算子によって、テキストフィールド Widget オブジェクトが初期化され、span コンテナが sprytextfield1 の ID を使用して静的 HTML コードからインタラクティブなページエレメントに変換されます。

Widget 内のエラーメッセージの span タグでは、CSS クラスが適用されています。このクラス (デフォルトでは display:none; に設定されています) は、エラーメッセージのスタイルと表示状態を制御し、対応する CSS ファイルである "SpryValidationTextField.css" に存在します。Widget がユーザーとのインタラクションの結果としてさまざまな状態を入力した場合、Spry は Widget のコンテナにさまざまなクラスを配置し、それによってエラーメッセージクラスに影響を与えます。

その他のエラーメッセージをテキストフィールド検査 Widget に追加するには、エラーメッセージのテキストを保持する span タグ (またはその他の種類のタグ) を作成します。次に、CSS クラスを適用することにより、Widget の状態に応じて、メッセージの表示、非表示を切り替えることができます。

対応する CSS ルールを "SpryValidationTextField.css" ファイル内に作成することにより、その他のエラーメッセージをテキストフィールド検査 Widget に追加できます。たとえば、状態の背景色を変更するには、スタイルシートで対応するルールを編集するか (まだルールが存在しない場合は) 新しいルールを追加します。

### テキストフィールド Widget 構造に使用されるさまざまなタグ

上の例では、span タグを使用して Widget の構造を作成しました。

```
Container SPAN
  INPUT type="text"
  Error message SPAN
```

ただし、Widget は、ほとんどすべてのコンテナタグを使用して作成できます。

```
Container DIV
  INPUT type="text"
  Error Message P
```

Spry では ( タグ自体ではなく ) タグ ID を使用して Widget が作成されます。Spry では、エラーメッセージも、エラーメッセージを含めるために使用される実際のタグとは関係のない CSS コードを使用して表示されます。

Widget コンストラクタに渡された ID は特定の HTML エlement を識別します。コンストラクタはこの Element を検出し、識別されたコンテナ内で対応する input タグを見つけます。コンストラクタに渡された ID が ( コンテナタグではなく ) input タグの ID である場合、コンストラクタは検査トリガを直接 input タグに添付します。ただし、コンテナタグが存在しない場合は、Widget はエラーメッセージを表示できず、さまざまな検査状態によって input タグ Element の外観 ( 背景色など ) だけが変更されます。

**注意:** 複数の INPUT タグは、同じ HTML Widget コンテナの内側では動作しません。各テキストフィールドは、独自の Widget である必要があります。

## テキストフィールド検査 Widget の CSS コード

"SpryValidationTextField.css" ファイルには、テキストフィールド検査 Widget とそのエラーメッセージのスタイルを設定するルールが含まれます。このルールを編集して、Widget およびエラーメッセージの外観と印象に関するスタイルを設定できます。CSS ファイル内のルールの名前は、Widget の HTML コードで指定されたクラスの名前に対応します。

"SpryValidationTextField.css" ファイルの CSS コードを次に示します。

```
/*Text Field styling classes*/
.textfieldRequiredMsg,
.textfieldInvalidFormatMsg,
.textfieldMinValueMsg,
.textfieldMaxValueMsg,
.textfieldMinCharsMsg,
.textfieldMaxCharsMsg,
.textfieldValidMsg {
  display: none;
}
.textfieldRequiredState .textfieldRequiredMsg,
.textfieldInvalidFormatState .textfieldInvalidFormatMsg,
.textfieldMinValueState .textfieldMinValueMsg,
.textfieldMaxValueState .textfieldMaxValueMsg,
.textfieldMinCharsState .textfieldMinCharsMsg,
.textfieldMaxCharsState .textfieldMaxCharsMsg {
  display: inline;
  color: #CC3333;
  border: 1px solid #CC3333;
}
.textfieldValidState input, input.textfieldValidState {
  background-color: #B8F5B1;
}
input.textfieldRequiredState, .textfieldRequiredState input,
input.textfieldInvalidFormatState, .textfieldInvalidFormatState input,
input.textfieldMinValueState, .textfieldMinValueState input,
input.textfieldMaxValueState, .textfieldMaxValueState input,
input.textfieldMinCharsState, .textfieldMinCharsState input,
input.textfieldMaxCharsState, .textfieldMaxCharsState input {
  background-color: #FF9F9F;
}
.textfieldFocusState input, input.textfieldFocusState {
  background-color: #FFFCC;
}
.textfieldFlashText input, input.textfieldFlashText {
  color: red !important;
}
```

"SpryValidationTextField.css" ファイルには、特定のルールに対するコードと目的を説明するさまざまなコメントも含まれます。詳細については、ファイル内のコメントを参照してください。

## テキストフィールド検査 Widget の挿入

1 "SpryValidationTextField.js" ファイルを検索して、Web サイトに追加します。"SpryValidationTextField.js" ファイルは、Adobe Labs からダウンロードした Spry ディレクトリにある `widgets/textfieldvalidation` ディレクトリ内で検索できます。詳細については、3 ページの「ファイルの準備」を参照してください。

たとえば、Web サイトのルートフォルダ内に "**SpryAssets**" という名前のフォルダを作成し、このフォルダに "SpryValidationTextField.js" ファイルを移動します。"SpryValidationTextField.js" ファイルには、テキストフィールド Widget をインタラクティブにするために必要な情報がすべて含まれます。

2 "SpryValidationTextField.css" ファイルを検索して、Web サイトに追加します。このファイルの追加先として、メインディレクトリである `SpryAssets` ディレクトリ、または CSS ディレクトリがある場合はそのディレクトリを選択できます。

3 テキストフィールド Widget の追加先となる Web ページを開き、このページのヘッドタグに次の script タグを挿入してこのページに "SpryValidationTextField.js" ファイルをリンクします。

```
<script src="SpryAssets/SpryValidationTextField.js" type="text/javascript"></script>
```

"SpryValidationTextField.js" ファイルへのファイルパスが正しいことを確認します。このパスは、Web サイト内のどこにファイルを配置したかによって異なります。

4 ページのヘッドタグに次の link タグを挿入して、Web ページに "SpryValidationTextField.css" ファイルをリンクします。

```
<link href="SpryAssets/SpryValidationTextField.css" rel="stylesheet" type="text/css" />
```

"SpryValidationTextField.css" ファイルへのファイルパスが正しいことを確認します。このパスは、Web サイト内のどこにファイルを配置したかによって異なります。

5 テキストフィールドをページに追加し、名前と一意の ID を指定します。

```
<input type="text" name="mytextfield" id="mytextfield"/>
```

6 テキストフィールドの周囲にコンテナを追加するには、`span` タグを `input` タグの周囲に挿入し、次のように一意の ID を Widget に割り当てます。

```
<span id="sprytextfield1">
  <input type="text" name="mytextfield" id="mytextfield"/>
</span>
```

7 次の script ブロックを Widget に対応する HTML コードの後に挿入して、テキストフィールドオブジェクトを初期化します。

```
<script type="text/javascript">
  var sprytextfield1 = new Spry.Widget.ValidationTextField("sprytextfield1");
</script>
```

`new JavaScript` 演算子によって、テキストフィールド Widget オブジェクトが初期化され、`span` タグコンテンツが `sprytextfield1` の ID を使用して静的 HTML コードからインタラクティブなテキストフィールドオブジェクトに変換されます。`Spry.Widget.ValidationTextField` メソッドは、テキストフィールドオブジェクトを作成する Spry フレームワークのコンストラクタです。このオブジェクトを初期化するために必要な情報は、この手順の開始時にリンクした `SpryValidationTextField.js` JavaScript ライブラリに含まれています。

テキストフィールドのコンテナ `span` タグの ID が `Spry.Widgets.ValidationTextField` メソッドで指定した ID パラメータと一致することを確認します。JavaScript の呼び出しが Widget に対応する HTML コードの後に行われることを確認します。

8 ページを保存します。

コード全体を次に示します。

```
<head>
...
<script src="SpryAssets/SpryValidationTextField.js" type="text/javascript"></script>
<link href="SpryAssets/SpryValidationTextField.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <span id="sprytextfield1">
    <input type="text" name="mytextfield" id="mytextfield" />
  </span>
<script type="text/javascript">
  var sprytextfield1 = new Spry.Widget.ValidationTextField("sprytextfield1");
</script>
</body>
```

## エラーメッセージの表示

❖ エラーメッセージを表示するための `span` タグ (またはその他の種類のタグ) を作成し、次のように適切なクラスを割り当てます。

```
<span id="sprytextfield1">
  <input type="text" name="mytextfield" id="mytextfield" />
  <span class="textfieldRequiredMsg">Please enter a description</span>
</span>
```

`textfieldRequiredMsg` ルールは `"SpryValidationTextField.css"` ファイル内に配置され、デフォルトでは `display:none` に設定されています。ユーザーとのインタラクションの中で `Widget` によって別の状態が入力されると、`Spry` により、この `Widget` のコンテナに適切なクラス (状態クラス) が適用されます。このアクションは、エラーメッセージクラスに影響を与えるので、結果として、エラーメッセージの外観に影響します。

たとえば、`"SpryValidationTextField.css"` ファイルの CSS ルールの一部は、次のようになります。

```
.textfieldRequiredMsg,
.textfieldInvalidFormatMsg,
.textfieldMinValueMsg,
.textfieldMaxValueMsg,
.textfieldMinCharsMsg,
.textfieldMaxCharsMsg,
.textfieldValidMsg {
  display: none;
}
.textfieldRequiredState .textfieldRequiredMsg,
.textfieldInvalidFormatState .textfieldInvalidFormatMsg,
.textfieldMinValueState .textfieldMinValueMsg,
.textfieldMaxValueState .textfieldMaxValueMsg,
.textfieldMinCharsState .textfieldMinCharsMsg,
.textfieldMaxCharsState .textfieldMaxCharsMsg {
  display: inline;
  color: #CC3333;
  border: 1px solid #CC3333;
}
```

デフォルトでは、状態クラスは `Widget` コンテナに適用されていないため、ページがブラウザにロードされるときに、上の HTML コード例のエラーメッセージテキストに適用されているのは `textfieldRequiredMsg` クラスだけです。このルールのプロパティと値のペアは `display:none` なので、メッセージは非表示のままになります。ただし、ユーザーが必須テキストフィールドにテキストを入力しない場合は、次のように `Spry` によって `Widget` コンテナに適切なクラスが適用されます。

```
<span id="sprytextfield1" class="textfieldRequiredState">
  <input type="text" name="mytextfield" id="mytextfield" />
  <span class="textfieldRequiredMsg">Please enter a description</span>
</span>
```

上の CSS コードでは、コンテキストセレクタである `.textfieldRequiredState .textfieldRequiredMsg` を持つ状態ルールによって、エラーメッセージを非表示にするデフォルトのエラーメッセージルールが上書きされます。このため、Spry によって状態クラスが Widget コンテナに適用されると、状態ルールによって Widget の外観が決定され、エラーメッセージは 1 ピクセルの実線のボーダーに囲まれて赤でインラインで表示されます。

デフォルトのエラーメッセージクラスとその説明のリストを次に示します。これらのクラスを変更して、名前を変更することができます。その場合は、必ずコンテキストセレクタも変更してください。

| エラーメッセージクラス                             | 説明                                    |
|---|---------------------------------------|
| <code>.textfieldRequiredMsg</code>      | Widget が必須状態を入力すると、エラーメッセージを表示します。    |
| <code>.textfieldInvalidFormatMsg</code> | Widget が無効な状態を入力すると、エラーメッセージを表示します。   |
| <code>.textfieldMinValueMsg</code>      | Widget が最小値状態を入力すると、エラーメッセージを表示します。   |
| <code>.textfieldMaxValueMsg</code>      | Widget が最大値状態を入力すると、エラーメッセージを表示します。   |
| <code>.textfieldMinCharsMsg</code>      | Widget が最小文字数状態を入力すると、エラーメッセージを表示します。 |
| <code>.textfieldMaxCharsMsg</code>      | Widget が最大文字数状態を入力すると、エラーメッセージを表示します。 |
| <code>.textfieldValidMsg</code>         | Widget が有効な状態を入力すると、エラーメッセージを表示します。   |

**注意：**状態関連のクラスの名前は、Spry フレームワークの一部としてハードコーディングされているため、変更することはできません。

### 検査の種類とフォーマットの指定

テキストフィールド検査 Widget では、さまざまな検査の種類、フォーマット、およびその他のオプションを指定できます。たとえば、クレジットカード番号を入力するテキストフィールドにはクレジットカード検査を指定できます。

検査の種類、フォーマット、およびその他のオプションは、Widget コンストラクタのパラメータとして指定します。コンストラクタの最初のパラメータは Widget コンテナ ID です。その後に検査の種類を 2 番目のパラメータとして指定し、さらにオプションで 3 番目のパラメータを指定します。3 番目のパラメータは、2 番目のパラメータで設定した検査の種類に依存するオプションの JavaScript 配列です。したがって、2 番目のパラメータを設定しない場合、3 番目のパラメータは設定できません。

**注意：**検査の種類が不要の場合に 3 番目のパラメータを設定するには、検査の種類を "none" に設定できます。

次のコードは、テキストフィールド Widget コンストラクタにさまざまなパラメータを指定するための一般的な構文を示しています。

```
<script type="text/javascript">
  var sprytextfield1= new Spry.Widget.ValidationTextField("WidgetContainerID", "ValidationType",
  {option1:"value1", option2:"value2", ..});
</script>
```

### 検査の種類オプション

ほとんどの検査では、検査が行われるために標準のフォーマットが必要になります。たとえばテキストフィールドに整数検査を適用した場合は、テキストフィールドに数字が入力されない限り検査は行われません。ただし検査の種類によっては、テキストフィールドに入力できるフォーマットの種類を選択できます。

次の表は、使用できる検査の種類とそのフォーマットのリストです。

| 検査の種類                  | フォーマット  |
|------------------------|---|
| none                   | 特別なフォーマットは不要です。   |
| integer                | 数字のみを入力できます。  |
| email                  | @記号およびピリオド(.)を含み、ピリオドの前後に1文字以上の文字がある電子メールアドレスのみを入力できます。   |
| date                   | フォーマットはさまざまです。  |
| time                   | フォーマットはさまざまです。  |
| credit_card            | フォーマットはさまざまです。  |
| zip_code               | フォーマットはさまざまです。  |
| phone_number           | 米国またはカナダの(000) 000-0000 フォーマット、またはカスタムフォーマットで入力できます。  |
| social_security_number | デフォルトでは、社会保証番号を 000-00-0000 のフォーマットで入力できます。   |
| currency               | 1,000,000.00 または 1.000.000,00 のフォーマットで金額を入力できます。  |
| real                   | 整数(1 など)、浮動値(12.123 など)、科学的記数法による浮動値(e を 10 の累乗数としたときの 1.212e+12、1.221e-12 など)など、さまざまな種類の数字や科学的記数法を検査します。 |
| ip                     | IPv4 と IPv6 のいずれかまたは両方を検査します。   |
| url                    | http://xxx.xxx.xxx、https://xxx.xxx.xxx、または ftp://xxx.xxx.xxx のフォーマットで URL を入力できます。                        |
| custom                 | カスタム検証の種類とフォーマットを指定できます。  |

### 整数の検査

❖ テキストフィールドに整数値のみを入力できるように設定するには、次に示すように、「integer」をコンストラクタの2番目のパラメータの値として追加します。

```
<script type="text/javascript">
    var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "integer");
</script>
```

整数検査が設定されたテキストフィールドには、負の値と正の値の両方を入力できます。正の値のみを入力できるようにするには、allowNegative オプションを3番目のパラメータに追加し、その値を false に設定します。

```
<script type="text/javascript">
    var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "integer",
    {allowNegative:false});
</script>
```

次の表は、3番目のパラメータで指定できるその他の一般的なオプションと値を示しています。

| オプション               | 値  |
|---------------------|--|
| format              | 該当なし   |
| validateOn          | ["blur"]; ["change"]; またはその両方 (["blur", "change"]) |
| isRequired          | true (デフォルト); false                                |
| useCharacterMasking | false (デフォルト); true                                |
| minChars/maxChars   | 該当なし   |
| minValue/maxValue   | 整数値  |
| pattern             | 該当なし   |

### 電子メールの検査

❖ テキストフィールドに標準の電子メールフォーマットのみを入力できるように設定するには、次に示すように、"email" をコンストラクタの 2 番目のパラメータの値として追加します。

```
<script type="text/javascript">
    var sprytextfield1= new Spry.Widget.ValidationTextField("sprytextfield1", "email");
</script>
```

次の構文を使用して、3 番目のパラメータのオプションを設定することもできます。

```
<script type="text/javascript">
    var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "email",
    {option:value});
</script>
```

次の表は、3 番目のパラメータで指定できる一般的なオプションと値を示しています。

| オプション               | 値  |
|---------------------|--|
| format              | 該当なし   |
| validateOn          | ["blur"]; ["change"]; またはその両方 (["blur", "change"]) |
| isRequired          | true (デフォルト); false                                |
| useCharacterMasking | 該当なし   |
| minChars/maxChars   | 正の整数値  |
| minValue/maxValue   | 該当なし   |
| pattern             | 該当なし   |

### 日付の検査

❖ テキストフィールドに日付フォーマットを入力できるように設定するには、次に示すように、"date" をコンストラクタの 2 番目のパラメータの値として追加します。

```
<script type="text/javascript">
    var sprytextfield1= new Spry.Widget.ValidationTextField("sprytextfield1", "date");
</script>
```

デフォルトの日付フォーマットは "mm/dd/yy" (米国の日付フォーマット) です。ただし、format オプションを使用すると、他の多数の日付フォーマットを 3 番目のパラメータで設定できます。以下に例を示します。

```
<script type="text/javascript">
    var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "date", {format:"yyyy-mm-dd"});
</script>
```

次の表は、3 番目のパラメータで指定できるフォーマットオプションの完全なリスト、およびその他の一般的なオプションと値を示しています。

| オプション               | 値   |
|---------------------|---|
| format              | "mm/dd/yy" (デフォルト); "mm/dd/yyyy"; "dd/mm/yyyy"; "dd/mm/yy"; "yy/mm/dd"; "yyyy/mm/dd"; "mm-dd-yy"; "dd-mm-yy"; "yyyy-mm-dd"; "mm.dd.yyyy"; "dd.mm.yyyy"; |
| validateOn          | ["blur"]; ["change"]; またはその両方 (["blur", "change"])  |
| isRequired          | true (デフォルト); false   |
| useCharacterMasking | false (デフォルト); true   |
| minChars/maxChars   | 該当なし  |
| minValue/maxValue   | 指定されたフォーマットの日付値   |
| pattern             | 該当なし  |

### 時刻の検査

❖ テキストフィールドに時刻フォーマットを入力できるように設定するには、次に示すように、"time" をコンストラクタの 2 番目のパラメータの値として追加します。

```
<script type="text/javascript">
    var sprytextfield1= new Spry.Widget.ValidationTextField("sprytextfield1", "time");
</script>
```

デフォルトの時刻フォーマットは "HH:mm" (時間および分) です。ただし、format オプションを使用すると、他の多数の時刻フォーマットを 3 番目のパラメータで設定できます。以下に例を示します。

```
<script type="text/javascript">
    var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "time",
    {format:"HH:mm:ss"});
</script>
```

次の表は、3 番目のパラメータで指定できるフォーマットオプションの完全なリスト、およびその他の一般的なオプションと値を示しています。

| オプション               | 値  |
|---------------------|--|
| format              | "HH:mm" (デフォルト); "HH:mm:ss"; "hh:mm tt"; "hh:mm:ss tt"; "hh:mm t"; "hh:mm:ss t"; ("tt" は am/pm フォーマットを表し、"t" は a/p フォーマットを表します。) |
| validateOn          | ["blur"]; ["change"]; またはその両方 (["blur", "change"])   |
| isRequired          | true (デフォルト); false  |
| useCharacterMasking | false (デフォルト); true  |
| minChars/maxChars   | 該当なし   |
| minValue/maxValue   | 指定されたフォーマットの時刻値  |
| pattern             | 該当なし   |

### クレジットカードの検査

❖ テキストフィールドにクレジットカードフォーマットを入力できるように設定するには、次に示すように、"credit\_card" をコンストラクタの 2 番目のパラメータの値として追加します。

```
<script type="text/javascript">
    var sprytextfield1= new Spry.Widget.ValidationTextField("sprytextfield1", "credit_card");
</script>
```

デフォルトでは、すべての種類のクレジットカードが検査されます。ただし、format オプションを使用すると、特定のクレジットカードフォーマットを 3 番目のパラメータで指定できます。以下に例を示します。

```
<script type="text/javascript">
    var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "credit_card",
    {format:"visa"});
</script>
```

次の表は、3 番目のパラメータで指定できるフォーマットオプションの完全なリスト、およびその他の一般的なオプションと値を示しています。

| オプション               | 値  |
|---------------------|--|
| format              | フォーマットなし (デフォルト); "visa"; "mastercard"; "amex"; "discover"; "dinersclub" |
| validateOn          | ["blur"]; ["change"]; またはその両方 (["blur", "change"])                       |
| isRequired          | true (デフォルト); false  |
| useCharacterMasking | false (デフォルト); true  |

| オプション             | 値    |
|-------------------|------|
| minChars/maxChars | 該当なし |
| minValue/maxValue | 該当なし |
| pattern           | 該当なし |

### 郵便番号の検査

❖ テキストフィールドに郵便番号フォーマットを入力できるように設定するには、次に示すように、"zip\_code" をコンストラクタの 2 番目のパラメータの値として追加します。

```
<script type="text/javascript">
    var sprytextfield1= new Spry.Widget.ValidationTextField("sprytextfield1", "zip_code");
</script>
```

デフォルトのフォーマットは、米国の 5 桁の郵便番号フォーマットです。ただし、format オプションを使用すると、他の多数のフォーマットを 3 番目のパラメータで指定できます。以下に例を示します。

```
<script type="text/javascript">
    var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "zip_code",
    {format:"zip_uk"});
</script>
```

次の表は、3 番目のパラメータで指定できるフォーマットオプションの完全なリスト、およびその他の一般的なオプションと値を示しています。

| オプション               | 値  |
|---------------------|--|
| format              | "zip_us5" ( デフォルト ); "zip_us9"; "zip_uk"; "zip_canada"; "zip_custom" |
| validateOn          | ["blur"]; ["change"]; またはその両方 (["blur", "change"])                   |
| isRequired          | true ( デフォルト ); false  |
| useCharacterMasking | false ( デフォルト ); true  |
| minChars/maxChars   | 該当なし   |
| minValue/maxValue   | 該当なし   |
| pattern             | カスタム郵便番号のパターンです。format に "zip_custom" を指定する場合に使用します。                 |

"zip\_custom" フォーマットを使用すると、カスタマイズした独自のパターンを郵便番号フォーマットとして指定できます。"zip\_custom" をフォーマットとして指定した後、パターンオプションを 3 番目のパラメータに追加してフォーマットを定義します。たとえば、3 桁の数字の後に他の数字や大文字と小文字が区別されるアルファベット文字が続く郵便番号の検査が必要になる場合があります。カスタムパターンを指定するには、コンストラクタの 3 番目のパラメータで、format オプションの後に pattern オプションを追加します。以下に例を示します。

```
<script type="text/javascript">
    var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "zip_code",
    {format:"zip_custom", pattern:"000 00AA"});
</script>
```

次の表は、カスタムパターンに使用する値の完全なリストです。

| 値        | 説明                      |
|----------|-------------------------|
| "0"      | 0 ~ 9 の整数です。            |
| "A"      | 大文字のアルファベットです。          |
| "a"      | 小文字のアルファベットです。          |
| "B"; "b" | 大文字と小文字を区別しないアルファベットです。 |
| "X"      | 大文字の英数字です。              |

| 値        | 説明                  |
|----------|---------------------|
| "x"      | 小文字の英数字です。          |
| "Y"; "y" | 大文字と小文字を区別しない英数字です。 |
| "?"      | 任意の文字です。            |

"A"、"a"、"X"、および"x"のカスタムパターン文字では、大文字と小文字が区別されます。これらの文字を使用する場合、ユーザーが大文字と小文字を誤って入力しても、正しい文字に自動的に変換されます。

### 電話番号の検査

❖ テキストフィールドに電話番号フォーマットを入力できるように設定するには、次に示すように、"phone\_number" をコンストラクタの2番目のパラメータの値として追加します。

```
<script type="text/javascript">
    var sprytextfield1= new Spry.Widget.ValidationTextField("sprytextfield1", "phone_number");
</script>
```

デフォルトのフォーマットは、米国の地域コードと電話番号のフォーマットである (000) 000-0000 です。ただし、"phone\_custom" オプションと "pattern" オプションを使用すると、カスタムフォーマットを3番目のパラメータで指定できます。以下に例を示します。

```
<script type="text/javascript">
    var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "zip_code",
    {format:"phone_custom" , pattern:"00 0000 A"});
</script>
```

次の表は、カスタムパターンに使用する値の完全なリストです。

| 値        | 説明                      |
|----------|-------------------------|
| "0"      | 0～9の整数です。               |
| "A"      | 大文字のアルファベットです。          |
| "a"      | 小文字のアルファベットです。          |
| "B"; "b" | 大文字と小文字を区別しないアルファベットです。 |
| "X"      | 大文字の英数字です。              |
| "x"      | 小文字の英数字です。              |
| "Y"; "y" | 大文字と小文字を区別しない英数字です。     |
| "?"      | 任意の文字です。                |

"A"、"a"、"X"、および"x"のカスタムパターン文字では、大文字と小文字が区別されます。これらの文字を使用する場合、ユーザーが大文字と小文字を誤って入力しても、正しい文字に自動的に変換されます。

次の表は、3番目のパラメータで指定できるその他の一般的なオプションと値を示しています。

| オプション               | 値  |
|---------------------|--|
| format              | "phone_number" (デフォルト); "phone_custom"                 |
| validateOn          | ["blur"]; ["change"]; またはその両方 (["blur", "change"])     |
| isRequired          | true (デフォルト); false                                    |
| useCharacterMasking | false (デフォルト); true                                    |
| minChars/maxChars   | 該当なし   |
| minValue/maxValue   | 該当なし   |
| pattern             | カスタム電話番号のパターンです。format に "phone_custom" を指定する場合に使用します。 |

### 社会保障番号の検査

❖ テキストフィールドに社会保障番号フォーマットを入力できるように設定するには、次に示すように、"social\_security\_number" をコンストラクタの 2 番目のパラメータの値として追加します。

```
<script type="text/javascript">
    var sprytextfield1= new Spry.Widget.ValidationTextField("sprytextfield1", "social_security number");
</script>
```

デフォルトのフォーマットは、米国のダッシュ付き社会保障番号である 000-00-0000 です。ただし、pattern オプションを使用すると、カスタムフォーマットを 3 番目のパラメータで指定できます。以下に例を示します。

```
<script type="text/javascript">
    var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1",
    "social_security_number", {format:"custom" pattern:"00 0000 A"});
</script>
```

次の表は、カスタムパターンに使用する値の完全なリストです。

| 値       | 説明                      |
|---------|-------------------------|
| "0"     | 0～9の整数です。               |
| "A"     | 大文字のアルファベットです。          |
| "a"     | 小文字のアルファベットです。          |
| "B";"b" | 大文字と小文字を区別しないアルファベットです。 |
| "X"     | 大文字の英数字です。              |
| "x"     | 小文字の英数字です。              |
| "Y";"y" | 大文字と小文字を区別しない英数字です。     |
| "?"     | 任意の文字です。                |

"A"、"a"、"X"、および"x"のカスタムパターン文字では、大文字と小文字が区別されます。これらの文字を使用する場合、ユーザーが大文字と小文字を誤って入力しても、正しい文字に自動的に変換されます。

次の表は、3 番目のパラメータで指定できるその他の一般的なオプションを示しています。

| オプション               | 値   |
|---------------------|---|
| format              | 該当なし  |
| validateOn          | ["blur"];["change"];またはその両方 (["blur","change"]) |
| isRequired          | true (デフォルト);false                              |
| useCharacterMasking | false (デフォルト);true                              |
| minChars/maxChars   | 該当なし  |
| minValue/maxValue   | 該当なし  |
| pattern             | カスタム社会保障番号のパターンです。                              |

### 通貨の検査

❖ テキストフィールドに通貨フォーマットを入力できるように設定するには、次に示すように、"currency" をコンストラクタの 2 番目のパラメータの値として追加します。

```
<script type="text/javascript">
    var sprytextfield1= new Spry.Widget.ValidationTextField("sprytextfield1", "currency");
</script>
```

デフォルトのフォーマットは、米国の通貨フォーマットである 1,000.00 です。ただし、次に示すように、"dot\_comma" フォーマット (1,000,00) を 3 番目のパラメータで指定できます。

```
<script type="text/javascript">
  var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "currency",
  {format:"dot_comma"});
</script>
```

いずれの場合も、3桁ごとの区切り文字と小数部(1,000.00の「.00」)は必須ではありません。

次の表は、3番目のパラメータで指定できるその他の一般的なオプションを示しています。

| オプション               | 値  |
|---------------------|--|
| format              | "comma_dot" (デフォルト); "dot_comma"                   |
| validateOn          | ["blur"]; ["change"]; またはその両方 (["blur", "change"]) |
| isRequired          | true (デフォルト); false                                |
| useCharacterMasking | false (デフォルト); true                                |
| minChars/maxChars   | 該当なし   |
| minValue/maxValue   | 小数点以下2桁の数値   |
| pattern             | 該当なし   |

### 実数および科学的記数法の検査

❖ テキストフィールドに実数および科学的記数法を入力できるように設定するには、次に示すように、"real" をコンストラクタの2番目のパラメータの値として追加します。

```
<script type="text/javascript">
  var sprytextfield1= new Spry.Widget.ValidationTextField("sprytextfield1", "real");
</script>
```

テキストフィールドでは、1.231e10などの科学的記数法の実数が検査されます。

次の表は、3番目のパラメータで指定できるその他の一般的なオプションを示しています。

| オプション               | 値  |
|---------------------|--|
| format              | 該当なし   |
| validateOn          | ["blur"]; ["change"]; またはその両方 (["blur", "change"]) |
| isRequired          | true (デフォルト); false                                |
| useCharacterMasking | false (デフォルト); true                                |
| minChars/maxChars   | 該当なし   |
| minValue/maxValue   | 小数を含む数値  |
| pattern             | 該当なし   |

### IPアドレスの検査

❖ テキストフィールドでIPアドレスを検査するように設定するには、次に示すように、"ip" をコンストラクタの2番目のパラメータとして追加します。

```
<script type="text/javascript">
  var sprytextfield1= new Spry.Widget.ValidationTextField("sprytextfield1", "ip");
</script>
```

デフォルトで入力できるIPアドレスフォーマットはIPv4です。ただし、formatオプションを使用すると、その他のIPアドレスフォーマットを3番目のパラメータで設定できます。以下に例を示します。

```
<script type="text/javascript">
  var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "ip",
  {format:"ipv6"});
</script>
```

次の表は、3番目のパラメータで指定できるフォーマットオプションとその他の一般的なオプションを示しています。

| オプション               | 値  |
|---------------------|--|
| format              | "ipv4" (デフォルト); "ipv6"; "ipv4_ipv6"                |
| validateOn          | ["blur"]; ["change"]; またはその両方 (["blur", "change"]) |
| isRequired          | true (デフォルト); false                                |
| useCharacterMasking | false (デフォルト); true                                |
| minChars/maxChars   | 該当なし   |
| minValue/maxValue   | 該当なし   |
| pattern             | 該当なし   |

### URL の検査

❖ テキストフィールドに URL 値のみを入力できるように設定するには、次に示すように、"url" をコンストラクタの 2 番目のパラメータの値として追加します。

```
<script type="text/javascript">
  var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "url");
</script>
```

URL 検査が設定されたテキストフィールドには、HTTP、HTTPS、および FTP の URL 値を入力できます。

次の表は、3 番目のパラメータで指定できるその他の一般的なオプションを示しています。

| オプション               | 値  |
|---------------------|--|
| format              | 該当なし   |
| validateOn          | ["blur"]; ["change"]; またはその両方 (["blur", "change"]) |
| isRequired          | true (デフォルト); false                                |
| useCharacterMasking | 該当なし   |
| minChars/maxChars   | 正の整数値  |
| minValue/maxValue   | 該当なし   |
| pattern             | 該当なし   |

### カスタムフォーマットの検査

❖ テキストフィールドに任意の種類のカスタムフォーマットを入力できるように設定するには、次に示すように、"custom" を 2 番目のパラメータの値として指定し、pattern オプションを 3 番目のパラメータに追加します。

```
<script type="text/javascript">
  var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "custom", {pattern:"0000 AX"});
</script>
```

次の表は、カスタムパターンに使用する値の完全なリストです。

| 値        | 説明                      |
|----------|-------------------------|
| "0"      | 0～9の整数です。               |
| "A"      | 大文字のアルファベットです。          |
| "a"      | 小文字のアルファベットです。          |
| "B"; "b" | 大文字と小文字を区別しないアルファベットです。 |
| "X"      | 大文字の英数字です。              |

| 値        | 説明                  |
|----------|---------------------|
| "x"      | 小文字の英数字です。          |
| "Y"; "y" | 大文字と小文字を区別しない英数字です。 |
| "?"      | 任意の文字です。            |

"A"、"a"、"X"、および"x"のカスタムパターン文字では、大文字と小文字が区別されます。これらの文字を使用する場合、ユーザーが大文字と小文字を誤って入力しても、正しい文字に自動的に変換されます。

### 検査を実行するタイミングの指定

デフォルトでは、テキストフィールド検査 Widget は、ユーザーが送信ボタンをクリックしたときに検査します。ただし、その他に、blur または change の 2 つのオプションを設定することもできます。validateOn:["blur"] パラメータを指定すると、Widget は、ユーザーがテキストフィールドの外側をクリックしたときに常に検査します。validateOn:["change"] パラメータを指定すると、Widget は、ユーザーがテキストフィールド内のテキストを変更したときに検査します。

❖ 検査を実行するタイミングを指定するには、次のように、validateOn パラメータをコンストラクタに追加します。

```
<script type="text/javascript">
  var sprytextfield1 = new Spry.Widget.ValidationTextField("sprytextfield1", "none",
  {validateOn:["blur"]});
</script>
```

validateOn パラメータに値を 1 つしか指定しない場合は、簡単にするため、角カッコを省略できます (たとえば、validateOn:"blur")。ただし、両方のパラメータを指定する場合は (validateOn:["blur","change"])、シンタックスに角カッコが必要です。

**注意:** 上の例では、2 番目のパラメータは "none" に設定されていますが、使用可能な検査の種類のいずれかに設定することも簡単にできます (たとえば、"phone" や "credit\_card")。詳細については、44 ページの「検査の種類とフォーマットの指定」を参照してください。

### 最小および最大文字数の指定

このオプションは、"none"、"integer"、"email"、および"url"の検査の種類でのみ使用できます。

minChars オプションの最小文字数は、テキストフィールドが必須ではない場合は適用されません。

❖ 最小文字数または最大文字数を指定するには、次のように、コンストラクタに minChars プロパティまたは maxChars プロパティ (あるいはその両方) と値を追加します。

```
<script type="text/javascript">
  var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "validation_type",
  {minChars:value, maxChars:value});
</script>
```

**注意:** 検査の種類が必要ない場合は、検査の種類を "none" に設定できます。詳細については、44 ページの「検査の種類とフォーマットの指定」を参照してください。

### 最小値および最大値の指定

このオプションは、"integer"、"date"、"time"、"currency"、および"real"の検査の種類でのみ使用できます。

❖ テキストフィールドの最小値または最大値を指定するには、次のように、コンストラクタに minValue プロパティまたは maxValue プロパティ (あるいはその両方) と値を追加します。

```
<script type="text/javascript">
  var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "validation_type",
  {minValue:value, maxValue:value});
</script>
```

## テキストフィールドの必須状態の変更

デフォルトでは、テキストフィールド検査 Widget を Web ページでパブリッシュすると、ユーザー入力が必須になります。ただし、テキストフィールドの入力をオプションにすることもできます。

❖ テキストフィールドの必須状態を変更するには、次のように `isRequired` プロパティをコンストラクタに追加し、その値を `false` に設定します。

```
<script type="text/javascript">
  var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "none",
  {isRequired:false});
</script>
```

**注意：**上の例では、2 番目のパラメータは `"none"` に設定されていますが、使用可能な検査の種類の内いずれかに設定することも簡単にできます (たとえば、`"phone"` や `"credit_card"`)。詳細については、44 ページの「検査の種類とフォーマットの指定」を参照してください。

## テキストフィールドのヒントの作成

テキストフィールドのフォーマットは種類が多いため、入力フォーマットをヒントで知らせるとユーザーに便利です。たとえば、電話番号検査が設定されたテキストフィールドは、デフォルトでは `(000) 000-0000` の形式の電話番号を受け入れます。ユーザーがブラウザでページを読み込んだときにテキストフィールドに正しいフォーマットが表示されるように、このサンプル電話番号をヒントとして入力することができます。

このオプションは、検査の種類に関係なく常に指定できます。必要な検査の種類がない場合は、検査の種類として `"none"` を指定してください。ユーザーがテキストフィールド内をクリックすると、ヒントは非表示になります。ユーザーがテキストフィールドの外側をクリックすると、フィールドに値が何も入力されていない場合は、再びヒントが表示されます。

❖ テキストフィールドのヒントを作成するには、次のように、`hint` プロパティとその値としてヒントのテキストをコンストラクタの 3 番目のパラメータに追加します。

```
<script type="text/javascript">
  var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "validation_type",
  {hint:"some hint text here"});
</script>
```

## 無効な文字のブロック

テキストフィールド検査 Widget にユーザーが無効な文字を入力するのを防止できます。たとえば、整数検査が設定された Widget にこのオプションを設定すると、ユーザーがこのテキストフィールドに文字を入力しようとしても何も表示されません。

このオプションでは、3 番目のパラメータの仕様が 2 番目のパラメータに応じて決まるので、検査の種類を指定する必要があります。必要な検査の種類がない場合は、検査の種類として `"none"` を指定してください。

このオプションは、古いブラウザでは機能しません。

❖ 無効な文字をブロックするには、次のように `useCharacterMasking` プロパティをコンストラクタに追加し、その値を `true` に設定します。

```
<script type="text/javascript">
  var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "validation_type",
  {useCharacterMasking:true});
</script>
```

## テキストフィールド検査 Widget のカスタマイズ

"`SpryValidationTextField.css`" ファイルには、テキストフィールド検査 Widget のデフォルトのスタイル設定が用意されています。ただし、該当する CSS ルールを変更することによって Widget をカスタマイズできます。

"`SpryValidationTextField.css`" ファイルの CSS ルールでは、Widget の HTML コードの関連するエレメントと同じクラス名が使用されるため、Widget とそのエラー状態に対応する CSS ルールを簡単に把握できます。

"`SpryValidationTextField.css`" ファイルは、カスタマイズを開始する前に Web サイトに配置しておく必要があります。詳細については、3 ページの「ファイルの準備」を参照してください。

### テキストフィールド検査 Widget のスタイル設定 (一般的な方法)

- 1 "SpryValidationTextField.css" ファイルを開きます。
- 2 Widget の変更する部分の CSS ルールを見つけます。たとえば、テキストフィールド Widget の必須状態の背景色を変更するには、"SpryValidationTextField.css" ファイルの .textfieldRequiredState ルールを編集します。
- 3 CSS を変更してファイルを保存します。

"SpryValidationTextField.css" ファイルには、特定のルールに対するコードと目的を説明するさまざまなコメントが含まれます。詳細については、ファイル内のコメントを参照してください。

### テキストフィールド検査 Widget のエラーメッセージのテキストのスタイル設定

テキストフィールド検査 Widget のエラーメッセージは、デフォルトでは、1 ピクセルの実線のボーダーで囲まれた赤いテキストで表示されます。

❖ テキストフィールド検査 Widget のエラーメッセージのテキストスタイルを変更するには、次の表を使用して適切な CSS ルールを見つけ、デフォルトのプロパティを変更するか、または独自のテキストスタイルプロパティおよび値を追加します。

| 変更するテキスト      | 関連する CSS ルール  | 変更する関連プロパティ                                |
|---------------|---|--|
| エラーメッセージのテキスト | .textfieldRequiredState .textfieldRequiredMsg、<br>.textfieldInvalidFormatState .textfieldInvalidFormatMsg、<br>.textfieldMinValueState .textfieldMinValueMsg、<br>.textfieldMaxValueState .textfieldMaxValueMsg、<br>.textfieldMinCharsState .textfieldMinCharsMsg、<br>.textfieldMaxCharsState .textfieldMaxCharsMsg | color: #CC3333; border: 1px solid #CC3333; |

### テキストフィールド検査 Widget の背景色の変更

❖ さまざまな状態のテキストフィールド検査 Widget の背景色を変更するには、次の表を使用して適切な CSS ルールを見つけ、デフォルトの背景色値を変更します。

| 変更する色                | 関連する CSS ルール   | 変更する関連プロパティ                |
|----------------------|--|----------------------------|
| 有効状態の Widget の背景色    | .textfieldValidState input、 input.textfieldValidState  | background-color: #B8F5B1; |
| 無効状態の Widget の背景色    | input.textfieldRequiredState、 .textfieldRequiredState input、<br>input.textfieldInvalidFormatState、<br>.textfieldInvalidFormatState input、<br>input.textfieldMinValueState、 .textfieldMinValueState<br>input、 input.textfieldMaxValueState、<br>.textfieldMaxValueState input、<br>input.textfieldMinCharsState、 .textfieldMinCharsState<br>input、 input.textfieldMaxCharsState、<br>.textfieldMaxCharsState input | background-color: #FF9F9F; |
| フォーカスがある Widget の背景色 | .textfieldFocusState input、 input.textfieldFocusState  | background-color: #FFFFCC; |

### カスタムパターンの指定

次の表は、カスタムパターンに使用する値の完全なリストです。

| 値        | 説明                      |
|----------|-------------------------|
| "0"      | 0～9の整数です。               |
| "A"      | 大文字のアルファベットです。          |
| "a"      | 小文字のアルファベットです。          |
| "B"; "b" | 大文字と小文字を区別しないアルファベットです。 |
| "X"      | 大文字の英数字です。              |

| 値        | 説明                  |
|----------|---------------------|
| "x"      | 小文字の英数字です。          |
| "Y"; "y" | 大文字と小文字を区別しない英数字です。 |
| "?"      | 任意の文字です。            |

これらの値を使用して、任意のフォーマットタイプのカスタムパターンを作成できます。たとえば、数字と大文字のアルファベットの組み合わせで構成されるカスタム社会保障番号を指定するには、コンストラクタの3番目のパラメータに次のカスタムパターンを指定します。

```
<script type="text/javascript">
    var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1",
"social_security_number", {format:"custom", pattern:"AA00AA"});
</script>
```

上のコードでは、UT99CW、AB87PP、GV44REなどの値を受け入れるテキストフィールドが作成されます。

**注意：**Dreamweaver ユーザーは、上の表に示されている値を使用して、プロパティインスペクタの [ パターン ] テキストボックスにカスタムパターンを入力するだけで済みます。たとえば、「AA00AA」と入力します。

**注意：**Spry の検査メカニズムでは、ASCII 文字のみをサポートしています。

### オートコンプリート文字の挿入

次の表は、カスタムパターンに使用する値の完全なリストです。

| 値        | 説明                      |
|----------|-------------------------|
| "0"      | 0～9の整数です。               |
| "A"      | 大文字のアルファベットです。          |
| "a"      | 小文字のアルファベットです。          |
| "B"; "b" | 大文字と小文字を区別しないアルファベットです。 |
| "X"      | 大文字の英数字です。              |
| "x"      | 小文字の英数字です。              |
| "Y"; "y" | 大文字と小文字を区別しない英数字です。     |
| "?"      | 任意の文字です。                |

バックスラッシュ (\) と上の表に示されている文字以外の文字 ( 文字や句読点など ) は、すべてオートコンプリート文字と見なされます。Spry では、これらの文字を適切なタイミングで挿入できます。たとえば、UT.99CW のような郵便番号を使用する場合、ユーザーが最初の2つの大文字を入力したら、自動的にピリオドを挿入することもできます。

❖ オートコンプリート文字を使用するには、次のように、フォーマットパターンに文字を挿入します。

```
<script type="text/javascript">
    var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "zip_code",
{format:"zip_custom", pattern:"AA.00AA"});
</script>
```

上のコードでは、UT.99CW、AB.87PP、GV.44REなどの値を受け入れるテキストフィールドが作成され、ユーザーがフィールドに最初の2つの大文字を入力すると、オートコンプリート文字として自動的にピリオドが表示されます。

Spry では、次の例のように、文字 ( 上の表の文字を除く ) をオートコンプリートすることもできます。

```
<script type="text/javascript">
    var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "zip_code",
{format:"zip_custom", pattern:"AA.00AAF3"});
</script>
```

上のコードでは、UT.99CWF3、AB.87PPF3、GV.44REF3 などの値を受け入れるテキストフィールドが作成されます。ユーザーがこのフィールドに最初の 2 つの大文字を入力すると、オートコンプリート文字としてピリオドが表示され、最後の 2 つの大文字を入力すると、F3 が表示されます。

上の表に示されている特殊文字をオートコンプリート文字として使用するには、次の例のように、2 つのバックスラッシュ (\) で特殊文字をエスケープします。

```
<script type="text/javascript">
    var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "zip_code",
    {format:"zip_custom", pattern:"AA.00AA\\B3"});
</script>
```

上のコードでは、UT.99CWB3、AB.87PPB3、GV.44REB3 などの値を受け入れるテキストフィールドが作成されます。ユーザーがこのフィールドに最初の 2 つの大文字を入力すると、オートコンプリート文字としてピリオドが表示され、最後の 2 つの大文字を入力すると、B3 が表示されます。

バックスラッシュ (\) をオートコンプリートのシーケンスの一部として使用するには、次のように、バックスラッシュを二重にしてから、2 つのバックスラッシュを使用してそれをエスケープします。合計では、4 つのバックスラッシュ (\\\\) になります。

```
<script type="text/javascript">
    var textfieldwidget1 = new Spry.Widget.ValidationTextField("textfieldwidget1", "zip_code",
    {format:"zip_custom", pattern:"AA\\\\\\\\00AA"});
</script>
```

上のコードでは、UT\\99CW、AB\\87PP、GV\\44RE などの値を受け入れるテキストフィールドが作成され、ユーザーがフィールドに最初の 2 つの大文字を入力すると、オートコンプリート文字として自動的にバックスラッシュが表示されます。

### 状態関連のクラス名のカスタマイズ

CSS コード内のルールと HTML コード内のクラス名を変更することによって、エラーメッセージ関連のクラス名を独自のクラス名に置き換えることはできますが、ピヘイビアは Spry フレームワークの一部としてハードコーディングされているため、状態関連のクラス名を変更または置換することはできません。ただし、Widget コンストラクタの 3 番目のパラメータで新しい値を指定することによって、デフォルトの状態関連のクラス名を独自の名前で上書きすることができます。

❖ Widget の状態関連のクラス名を変更するには、次のように、上書きを行うオプションのいずれかを Widget コンストラクタの 3 番目のパラメータに追加し、カスタムクラス名を指定します。

```
<script type="text/javascript">
    var sprytextfield1 = new Spry.Widget.ValidationTextField("sprytextfield1", "none",
    {requiredClass:"required"});
</script>
```

次の表に、組み込みの状態関連のクラス名を上書きするために使用できるオプションのリストを示します。

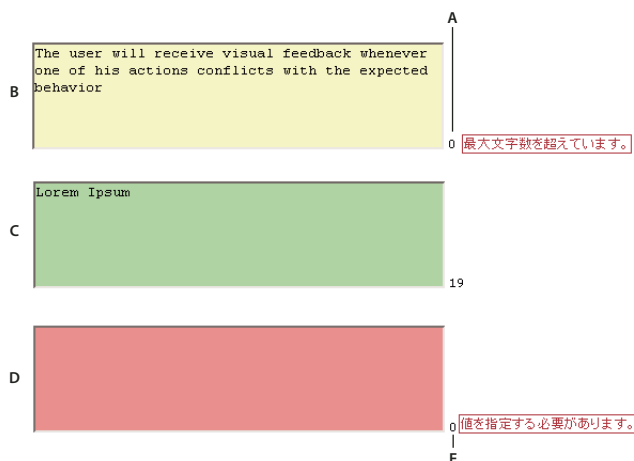
| オプション                   | 説明  |
|-------------------------|---|
| requiredClass           | "textfieldRequiredState" 組み込み値を上書きします。      |
| invalidFormatClass      | "textfieldInvalidFormatState" 組み込み値を上書きします。 |
| validClass              | "textfieldValidState" 組み込み値を上書きします。         |
| focusClass              | "textfieldFocusState" 組み込み値を上書きします。         |
| invalidFormatClass      | "textfieldInvalidFormatState" 組み込み値を上書きします。 |
| invalidRangeMinClass    | "textfieldMinValueState" 組み込み値を上書きします。      |
| invalidRangeMaxClass    | "textfieldMaxValueState" 組み込み値を上書きします。      |
| invalidCharsMinClass    | "textfieldMinCharsState" 組み込み値を上書きします。      |
| invalidCharsMaxClass    | "textfieldMaxCharsState" 組み込み値を上書きします。      |
| textfieldFlashTextClass | "textfieldFlashText" 組み込み値を上書きします。          |

## テキスト領域検査 Widget の操作

### テキスト領域検査 Widget の概要と構造

Spry テキスト領域検査 Widget は、ユーザーが文をいくつか入力したときに有効か無効かを表示するテキスト領域です。必須フィールドであるテキスト領域にユーザーが何もテキストを入力していない場合は、値が必須であることを示すメッセージが返されます。

さまざまな状態のテキスト領域検査 Widget の例を次に示します。



A. 入力可能な残り文字数カウンタ B. フォーカスがあるときのテキスト領域 Widget、最大文字数状態 C. フォーカスがあるときのテキスト領域 Widget、有効状態 D. テキスト領域 Widget、必須状態 E. 入力文字数カウンタ

テキスト領域検査 Widget には、有効、無効、必須などのいくつかの状態が含まれます。これらの状態のプロパティは、使用する検査結果に応じて、プロパティインスペクタで変更できます。検査が実行されるタイミングは、さまざまに設定できます。たとえば、ユーザーが Widget の外側をクリックしたとき、入力したとき、フォームを送信しようとしたときなどに設定できます。

**初期状態** ページがブラウザに読み込まれたとき、またはユーザーがフォームをリセットしたとき。

**フォーカス状態** ユーザーが Widget に挿入ポイントを置いたとき。

**有効状態** ユーザーが情報を正しく入力し、フォームを送信できる状態になったとき。

**必須状態** ユーザーがテキストを何も入力していないとき。

**最小文字数状態** ユーザーが入力した文字数がテキスト領域の最小文字数に満たないとき。

**最大文字数状態** ユーザーが入力した文字数がテキスト領域の最大文字数を超過しているとき。

ユーザーとのインタラクションの中で、テキスト領域検査 Widget によってこれらのいずれかの状態が入力されると、実行時に Spry フレームワークロジックにより、この Widget の HTML コンテナに特定の CSS クラスが適用されます。たとえば、ユーザーがフォームを送信しようとしたときにテキスト領域にテキストが入力されていない場合は、「値を指定する必要があります。」というエラーメッセージを表示するクラスが Widget に適用されます。エラーメッセージのスタイルと表示状態を制御するルールは、この Widget に対応する "SpryValidationTextarea.css" ファイルに存在します。

テキスト領域検査 Widget のデフォルト HTML コードは、通常はフォームの内部に配置され、コンテナ span タグで構成されます。このタグはテキスト領域の `textarea` タグを囲みます。テキスト領域検査 Widget の HTML コードには、ドキュメントのヘッド内とこの Widget の HTML コードの後の `script` タグも含まれます。

テキスト領域検査 Widget の HTML コードには、ドキュメントのヘッド内とこの Widget の HTML コードの後の `script` タグも含まれます。Widget コードの後の `script` タグは、テキスト領域をインタラクティブにする JavaScript オブジェクトを作成します。

テキスト領域検査 Widget の HTML コードを次に示します。

```
<head>
...
<!-- Link the Spry Validation Text Area JavaScript library -->
<script src="SpryAssets/SpryValidationTextarea.js" type="text/javascript"></script>
<!-- Link the CSS style sheet that styles the widget -->
<link href="SpryAssets/SpryValidationTextarea.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" name="form1" method="post" action="">
    <!-- Create the text area widget and assign a unique id-->
    <span id="sprytextarea1">
      <textarea name="textarea1" id="textarea1" cols="45" rows="5"></textarea>
      <!--Display an error message-->
      <span class="textareaRequiredMsg">A value is required.</span>
    </span>
  </form>
<!-- Initialize the Validation Text Area widget object-->
<script type="text/javascript">
var sprytextarea1 = new Spry.Widget.ValidationTextarea("sprytextarea1");
</script>
</body>
```

このコードでは、new JavaScript の演算子によって、テキスト領域 Widget オブジェクトが初期化され、span コンテンツが sprytextarea1 の ID を使用して静的 HTML コードからインタラクティブなページエレメントに変換されます。

Widget 内のエラーメッセージの span タグでは、CSS クラスが適用されています。このクラス ( デフォルトでは display:none; に設定されています ) は、エラーメッセージのスタイルと表示状態を制御し、対応する "SpryValidationTextarea.css" ファイルに存在します。Widget がユーザーとのインタラクションの結果としてさまざまな状態を入力した場合、Spry は Widget のコンテナにさまざまなクラスを配置し、それによってエラーメッセージクラスに影響を与えます。

その他のエラーメッセージをテキスト領域検査 Widget に追加するには、エラーメッセージのテキストを保持する span タグ ( またはその他の種類のタグ ) を作成します。次に、CSS クラスを適用することにより、Widget の状態に応じて、メッセージの表示、非表示を切り替えることができます。

デフォルトのテキスト領域検査 Widget の状態の外観は、"SpryValidationTextarea.css" ファイル内の対応する CSS ルールを編集することで変更できます。たとえば、状態の背景色を変更するには、スタイルシートで対応するルールを編集するか ( まだルールが存在しない場合は ) 新しいルールを追加します。

### テキスト領域 Widget 構造に使用されるさまざまなタグ

上の例では、span タグによって Widget の構造が作成されます。

```
Container SPAN
  TEXTAREA tag
  Error message SPAN
```

ただし、Widget は、ほとんどすべてのコンテナタグを使用して作成できます。

```
Container DIV
  TEXTAREA tag
  Error Message P
```

Spry では ( タグ自体ではなく ) タグ ID を使用して Widget が作成されます。Spry では、エラーメッセージも、エラーメッセージを含めるために使用される実際のタグとは関係のない CSS コードを使用して表示されます。

Widget コンストラクタに渡された ID は特定の HTML エレメントを識別します。コンストラクタはこのエレメントを検出し、識別されたコンテナ内で対応する textarea タグを見つけます。コンストラクタに渡された ID が ( コンテナタグではなく ) textarea タグの ID である場合、コンストラクタは検査トリガを直接 textarea タグに添付します。ただし、コンテナタグがない場合は、Widget はエラーメッセージを表示できず、さまざまな検査状態によって textarea タグエレメントの外観 ( 背景色など ) だけが変更されます。

**注意:** 複数の textarea タグは、同じ HTML Widget コンテナの内側では動作しません。各テキストフィールドは、独自の Widget である必要があります。

## テキスト領域検査 Widget の CSS コード

"SpryValidationTextarea.css" ファイルには、テキスト領域検査 Widget とそのエラーメッセージのスタイルを設定するルールが含まれます。このルールを編集して、Widget およびエラーメッセージの外観と印象に関するスタイルを設定できます。CSS ファイル内のルールの名前は、Widget の HTML コードで指定されたクラスの名前に対応します。

"SpryValidationTextarea.css" ファイルの CSS コードを次に示します。

```
/*Validation Textarea styling classes*/
.textareaRequiredMsg, .textareaMinCharsMsg, .textareaMaxCharsMsg, .textareaValidMsg {
    display:none;
}
.textareaRequiredState .textareaRequiredMsg,
.textareaMinCharsState .textareaMinCharsMsg,
.textareaMaxCharsState .textareaMaxCharsMsg{
    display: inline;
    color: #CC3333;
    border: 1px solid #CC3333;
}
.textareaValidState textarea, textarea.textareaValidState {
    background-color:#B8F5B1;
}
textarea.textareaRequiredState, .textareaRequiredState textarea, textarea.textareaMinCharsState,
.textareaMinCharsState textarea, textarea.textareaMaxCharsState, .textareaMaxCharsState textarea {
    background-color:#FF9F9F;
}
.textareaFocusState textarea, textarea.textareaFocusState {
    background-color:#FFFCC;
}
.textareaFlashState textarea, textarea.textareaFlashState{
    color:red !important;
}
```

CSS ファイルには、特定のルールに対するコードと目的を説明するさまざまなコメントも含まれます。詳細については、ファイル内のコメントを参照してください。

## テキスト領域検査 Widget の挿入

**1** "SpryValidationTextarea.js" ファイルを検索して、Web サイトに追加します。"SpryValidationTextarea.js" ファイルは、Adobe Labs からダウンロードした Spry ディレクトリにある widgets/textareavalidation ディレクトリ内で検索できます。詳細については、3 ページの「ファイルの準備」を参照してください。

たとえば、Web サイトのルートフォルダ内に **SpryAssets** という名前のフォルダを作成し、そのフォルダに "SpryValidationTextarea.js" ファイルをアップロードします。"SpryValidationTextarea.js" ファイルには、テキスト領域 Widget をインタラクティブにするために必要な情報がすべて含まれます。

**2** "SpryValidationTextarea.css" ファイルを検索して、Web サイトに追加します。このファイルの追加先として、メインディレクトリである SpryAssets ディレクトリ、または CSS ディレクトリがある場合はそのディレクトリを選択できます。

**3** テキスト領域 Widget の追加先となる Web ページを開き、このページのヘッドタグに次の script タグを挿入してこのページに "SpryValidationTextarea.js" ファイルをリンクします。

```
<script src="SpryAssets/SpryValidationTextarea.js" type="text/javascript"></script>
```

"SpryValidationTextarea.js" ファイルへのファイルパスが正しいことを確認します。このパスは、Web サイト内のどこにファイルを配置したかによって異なります。

**4** ページのヘッドタグに次の link タグを挿入して、Web ページに "SpryValidationTextarea.css" ファイルをリンクします。

```
<link href="SpryAssets/SpryValidationTextarea.css" rel="stylesheet" type="text/css" />
```

"SpryValidationTextarea.css" ファイルへのファイルパスが正しいことを確認します。このパスは、Web サイト内のどこにファイルを配置したかによって異なります。

5 テキスト領域をページに追加し、名前と一意の ID を指定します。

```
<textarea name="mytextarea" id="textarea"></textarea>
```

6 次のように `span` タグを `<textarea>` タグの周囲に挿入し、一意の ID を `Widget` に割り当てることによって、テキスト領域の周囲にコンテナを追加します。

```
<span id="sprytextarea1">  
  <textarea name="mytextarea"></textarea>  
</span>
```

7 次のスクリプトブロックを `Widget` に対応する HTML コードの後に挿入して、テキスト領域オブジェクトを初期化します。

```
<script type="text/javascript">  
  var sprytextarea1 = new Spry.Widget.ValidationTextarea("sprytextarea1");  
</script>
```

`new JavaScript` の演算子によって、テキスト領域 `Widget` オブジェクトが初期化され、`span` タグコンテンツが `sprytextarea1` の ID を使用して静的 HTML コードからインタラクティブなテキストフィールドオブジェクトに変換されます。`Spry.Widget.ValidationTextarea` メソッドは、テキスト領域オブジェクトを作成する `Spry` フレームワークのコンストラクタです。このオブジェクトを初期化するために必要な情報は、この手順の開始時にリンクした `JavaScript` ライブラリの `SpryValidationTextarea.js` に含まれています。

テキスト領域のコンテナ `span` タグの ID が `Spry.Widgets.ValidationTextarea` メソッドで指定した ID パラメータと一致することを確認します。`JavaScript` の呼び出しが `Widget` に対応する HTML コードの後に行われることを確認します。

8 ページを保存します。

コード全体を次に示します。

```
<head>  
  ...  
<script src="SpryAssets/SpryValidationTextarea.js" type="text/javascript"></script>  
<link href="SpryAssets/SpryValidationTextarea.css" rel="stylesheet" type="text/css" />  
</head>  
<body>  
  <form id="form1" name="form1" method="post" action="">  
    <span id="sprytextarea1">  
      <textarea name="textarea1" id="textarea1" cols="45" rows="5"></textarea>  
    </span>  
  </form>  
<script type="text/javascript">  
  var sprytextarea1 = new Spry.Widget.ValidationTextarea("sprytextarea1");  
</script>  
</body>
```

## エラーメッセージの表示

❖ エラーメッセージを表示するための `span` タグ (またはその他の種類のタグ) を作成し、次のように適切なクラスを割り当てます。

```
<span id="sprytextarea1">  
  <textarea name="textarea1" id="textarea1" cols="45" rows="5"></textarea>  
  <span class="textareaRequiredMsg">Please enter a description</span>  
</span>
```

`.textareaRequiredMsg` ルールは `"SpryValidationTextarea.css"` ファイルに配置され、デフォルトでは `display:none` に設定されています。ユーザーとのインタラクションの中で `Widget` によって別の状態が入力されると、`Spry` により、この `Widget` のコンテナに適切なクラス (状態クラス) が適用されます。このアクションは、エラーメッセージクラスに影響を与えるので、結果として、エラーメッセージの外観に影響します。

たとえば、`"SpryValidationTextarea.css"` ファイルの CSS の一部は、次のようになります。

```
.textareaRequiredMsg,
.textareaMinCharsMsg,
.textareaMaxCharsMsg,
.textareaValidMsg {
  display:none;
}
.textareaRequiredState .textareaRequiredMsg,
.textareaMinCharsState .textareaMinCharsMsg,
.textareaMaxCharsState .textareaMaxCharsMsg {
  display: inline;
  color: #CC3333;
  border: 1px solid #CC3333;
}
```

デフォルトでは、状態クラスは **Widget** コンテナに適用されていないため、ページがブラウザに読み込まれるときに、上の HTML コード例のエラーメッセージテキストに適用されているのは `.textareaRequiredMsg` クラスだけです。このルールのプロパティと値のペアは `display:none` なので、メッセージは非表示のままになります。ただし、ユーザーが必須テキスト領域にテキストを入力しない場合は、次のように **Spry** によって **Widget** コンテナに適切なクラスが適用されます。

```
<span id="sprytextarea1" class="textareaRequiredState">
  <input type="text" name="mytextarea" id="mytextarea" />
  <span class="textareaRequiredMsg">Please enter a description</span>
</span>
```

上のコードの CSS では、コンテキストセレクタである `.textareaRequiredState .textareaRequiredMsg` を持つ状態ルールによって、エラーメッセージテキストを非表示にするデフォルトのエラーメッセージルールが上書きされます。このため、**Spry** によって状態クラスが **Widget** コンテナに適用されると、状態ルールによって **Widget** の外観が決定され、エラーメッセージは 1 ピクセルの実線のボーダーに囲まれて赤でインラインで表示されます。

デフォルトのエラーメッセージクラスとその説明のリストを次に示します。これらのクラスを変更して、任意の名前に変更することができます。その場合は、必ずコンテキストセレクタも変更してください。

| エラーメッセージクラス                       | 説明                                    |
|-----------------------------------|---------------------------------------|
| <code>.textareaRequiredMsg</code> | Widget が必須状態を入力すると、エラーメッセージを表示します。    |
| <code>.textareaMinCharsMsg</code> | Widget が最小文字数状態を入力すると、エラーメッセージを表示します。 |
| <code>.textareaMaxCharsMsg</code> | Widget が最大文字数状態を入力すると、エラーメッセージを表示します。 |
| <code>.textareaValidMsg</code>    | Widget が有効な状態を入力すると、エラーメッセージを表示します。   |

**注意：**状態関連のクラスの名前は、**Spry** フレームワークの一部としてハードコーディングされているため、変更することはできません。

## 検査を実行するタイミングの指定

デフォルトでは、テキスト領域検査 **Widget** は、ユーザーが送信ボタンをクリックしたときに検査します。ただし、その他に、`blur` または `change` の 2 つのオプションを設定することもできます。`validateOn:["blur"]` パラメータを指定すると、**Widget** は、ユーザーがテキスト領域の外側をクリックしたときに常に検査します。`validateOn:["change"]` パラメータを指定すると、**Widget** は、ユーザーがテキスト領域内のテキストを変更したときに検査します。

❖ 検査を実行するタイミングを指定するには、次のように、`validateOn` パラメータをコンストラクタに追加します。

```
<script type="text/javascript">
  var sprytextarea1 = new Spry.Widget.ValidationTextarea("sprytextarea1", {validateOn:["blur"]});
</script>
```

`validateOn` パラメータに値を 1 つしか指定しない場合は、簡単にするため、角カッコを省略できます (たとえば、`validateOn:"blur"`)。ただし、両方のパラメータを指定する場合は (`validateOn:["blur","change"]`)、シンタックスに角カッコが必要です。

## 最小および最大文字数の指定

❖ 最小文字数または最大文字数を指定するには、次のように、コンストラクタに `minChars` プロパティまたは `maxChars` プロパティ (あるいはその両方) と値を追加します。

```
<script type="text/javascript">
    var textareawidget1 = new Spry.Widget.ValidationTextarea("textareawidget1", {minChars:value,
maxChars:value});
</script>
```

## 文字カウンタの追加

ユーザーが何文字入力したかを知ることができる文字カウンタ、またはそのテキスト領域の入力文字数が残り何文字かを知ることができる文字カウンタを追加できます。

**1** 次のように、`span` タグを `Widget` の `textarea` タグの後に追加して、一意の ID をタグに割り当てます。

```
<form id="form1" name="form1" method="post" action="">
    <span id="sprytextarea1">
        <textarea name="textarea1" id="textarea1" cols="45" rows="5"></textarea>
        <span id="my_counter_span"></span>
        <span class="textareaRequiredMsg">Maximum number of characters exceeded</span>
    </span>
</form>
<script type="text/javascript">
    var sprytextarea1 = new Spry.Widget.ValidationTextarea("sprytextarea1" {maxChars:100});
</script>
```

新しいタグは空のままにしておきます。後でユーザーがテキストを入力したときに、`Spry` によりタグのコンテンツが提供されます。

**注意:** カウンタタグは、`Widget` の HTML コンテナタグ内に配置する必要があります。

**2** 次のように、`counterType` オプションと値を `Widget` コンストラクタに追加します。

```
<form id="form1" name="form1" method="post" action="">
    <span id="sprytextarea1">
        <textarea name="textarea1" id="textarea1" cols="45" rows="5"></textarea>
        <span id="my_counter_span"></span>
        <span class="textareaRequiredMsg">Maximum number of characters exceeded</span>
    </span>
</form>
<script type="text/javascript">
    var sprytextarea1 = new Spry.Widget.ValidationTextarea("sprytextarea1" {maxChars:100,
counterType:"chars_remaining"});
</script>
```

`counterType` オプションでは、使用するカウンタのタイプを定義します。指定できる値は、`"chars_count"` または `"chars_remaining"` の 2 つです。`"chars_count"` 値を指定すると、カウンタはテキスト領域に入力された文字数をカウントします。`"chars_remaining"` 値を指定すると、カウンタは最大文字数が入力されるまでの残りの文字数を表示します。2 番目のオプションは、上の例のように、`maxChars` オプションと共に使用する必要があります。詳細については、63 ページの「最小および最大文字数の指定」を参照してください。

**3** 次のように、`counterId` オプションを `Widget` コンストラクタに追加して、カウンタ `span` タグに割り当てた一意の ID と同じ値を設定します。

```
<form id="form1" name="form1" method="post" action="">
    <span id="sprytextarea1">
        <textarea name="textarea1" id="textarea1" cols="45" rows="5"></textarea>
        <span id="my_counter_span"></span>
        <span class="textareaRequiredMsg">Maximum number of characters exceeded</span>
    </span>
</form>
<script type="text/javascript">
    var sprytextarea1 = new Spry.Widget.ValidationTextarea("sprytextarea1" {maxChars:100,
counterType:"chars_remaining", counterId:"my_counter_span"});
</script>
```

## テキスト領域の必須状態の変更

デフォルトでは、テキスト領域検査 Widget を Web ページでパブリッシュすると、ユーザー入力が必要になります。ただし、テキスト領域の入力をオプションにすることもできます。

❖ テキスト領域の必須状態を変更するには、次のように `isRequired` プロパティをコンストラクタに追加し、その値を `false` に設定します。

```
<script type="text/javascript">
  var textareawidget1 = new Spry.Widget.ValidationTextarea("textareawidget1", {isRequired:false});
</script>
```

## テキスト領域のヒントの作成

`hint` オプションを使用すると、どのようなテキストを入力する必要があるのかをユーザーに知らせるヒントを表示できます (「Enter your address here」など)。ヒントは、ページがブラウザに読み込まれたときに定義済みの値がなければ、テキスト領域に表示されます。

❖ テキスト領域のヒントを作成するには、次のように、`hint` プロパティとその値としてヒントのテキストをコンストラクタに追加します。

```
<script type="text/javascript">
  var textareawidget1 = new Spry.Widget.ValidationTextarea("textareawidget1", {hint:"Enter your address here"});
</script>
```

## 余分な文字のブロック

ユーザーが、テキスト領域検査 Widget の最大文字数を超える文字を入力するのを防止できます。たとえば、Widget で受け入れる最大文字数を 20 文字にするように `useCharacterMasking` オプションを設定すると、ユーザーはこのテキスト領域に 20 文字までしか入力できなくなります。

このオプションは、`maxChars` と共に使用します。詳細については、63 ページの「最小および最大文字数の指定」を参照してください。

❖ 余分な文字をブロックするには、次のように `useCharacterMasking` プロパティをコンストラクタに追加し、その値を `true` に設定します。

```
<script type="text/javascript">
  var textareawidget1 = new Spry.Widget.ValidationTextarea("textareawidget1", maxChars:20,
  {useCharacterMasking:true});
</script>
```

## テキスト領域検査 Widget のカスタマイズ

"SpryValidationTextarea.css" ファイルには、テキスト領域検査 Widget のデフォルトのスタイル設定が用意されています。ただし、該当する CSS ルールを変更することによって Widget をカスタマイズできます。

"SpryValidationTextarea.css" ファイルの CSS ルールでは、Widget の HTML コードの関連するエレメントと同じクラス名が使用されるため、Widget とそのエラー状態に対応する CSS ルールを簡単に把握できます。

"SpryValidationTextarea.css" ファイルは、カスタマイズを開始する前に Web サイトに配置しておく必要があります。詳細については、3 ページの「ファイルの準備」を参照してください。

### テキスト領域検査 Widget のスタイル設定 (一般的な方法)

- 1 "SpryValidationTextarea.css" ファイルを開きます。
- 2 Widget の変更する部分の CSS ルールを見つけます。たとえば、テキスト領域 Widget の必須状態の背景色を変更するには、"SpryValidationTextarea.css" ファイルの `textareaRequiredState` ルールを編集します。
- 3 CSS ルールを変更してファイルを保存します。

"SpryValidationTextarea.css" ファイルには、特定のルールに対するコードと目的を説明するさまざまなコメントが含まれます。詳細については、ファイル内のコメントを参照してください。

### テキスト領域検査 Widget のエラーメッセージのテキストのスタイル設定

テキスト領域検査 Widget のエラーメッセージは、デフォルトでは、1 ピクセルの実線のボーダーで囲まれた赤いテキストで表示されます。

❖ テキスト領域検査 Widget のエラーメッセージのテキストスタイルを変更するには、次の表を使用して適切な CSS ルールを見つけ、デフォルトのプロパティを変更するか、または独自のテキストスタイルプロパティおよび値を追加します。

| 変更するテキスト      | 関連する CSS ルール   | 変更する関連プロパティ                                |
|---------------|--|--|
| エラーメッセージのテキスト | .textareaRequiredState<br>.textareaRequiredMsg、<br>.textareaMinCharsState<br>.textareaMinCharsMsg、<br>.textareaMaxCharsState<br>.textareaMaxCharsMsg | color: #CC3333; border: 1px solid #CC3333; |

### テキスト領域検査 Widget の背景色の変更

❖ さまざまな状態のテキスト領域検査 Widget の背景色を変更するには、次の表を使用して適切な CSS ルールを見つけ、デフォルトの背景色値を変更します。

| 変更する背景色              | 関連する CSS ルール   | 変更する関連プロパティ                |
|----------------------|--|----------------------------|
| 有効状態の Widget の背景色    | .textareaValidState textarea、<br>textarea.textareaValidState   | background-color: #B8F5B1; |
| 無効状態の Widget の背景色    | textarea.textareaRequiredState、<br>.textareaRequiredState textarea、<br>textarea.textareaMinCharsState、<br>.textareaMinCharsState textarea、<br>textarea.textareaMaxCharsState、<br>.textareaMaxCharsState textarea | background-color: #FF9F9F; |
| フォーカスがある Widget の背景色 | .textareaFocusState textarea、<br>textarea.textareaFocusState   | background-color: #FFFFCC; |

### 状態関連のクラス名のカスタマイズ

CSS 内のルールと HTML コード内のクラス名を変更することによって、エラーメッセージ関連のクラス名を独自のクラス名に置き換えることはできますが、ビヘイビアは Spry フレームワークの一部としてハードコーディングされているため、状態関連のクラス名を変更または置換することはできません。ただし、Widget コンストラクタの 3 番目のパラメータで新しい値を指定することによって、デフォルトの状態関連のクラス名を独自の名前で上書きすることができます。

❖ Widget の状態関連のクラス名を変更するには、次のように、上書きを行うオプションのいずれかを Widget コンストラクタの 3 番目のパラメータに追加し、カスタムクラス名を指定します。

```
<script type="text/javascript">
    var sprytextareal = new Spry.Widget.ValidationTextarea("sprytextareal1", {requiredClass:"required"});
</script>
```

次の表に、組み込みの状態関連のクラス名を上書きするために使用できるオプションのリストを示します。

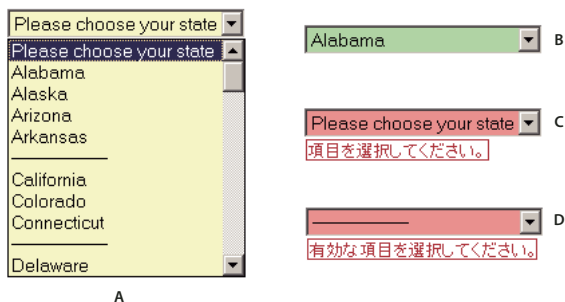
| オプション                  | 説明                                    |
|------------------------|---------------------------------------|
| requiredClass          | "textareaRequiredState" 組み込み値を上書きします。 |
| validClass             | "textareaValidState" 組み込み値を上書きします。    |
| focusClass             | "textareaFocusState" 組み込み値を上書きします。    |
| invalidCharsMinClass   | "textareaMinCharsState" 組み込み値を上書きします。 |
| invalidCharsMaxClass   | "textareaMaxCharsState" 組み込み値を上書きします。 |
| textareaFlashTextClass | "textareaFlashText" 組み込み値を上書きします。     |

## 選択検査 Widget の操作

### 選択検査 Widget の概要と構造

Spry 選択検査 Widget は、ユーザーが選択すると有効状態または無効状態が表示されるドロップダウンメニューです。たとえば、ある選択検査 Widget に含まれるリストが、横線で区切ってグループ化されているとします。このときユーザーが誤って区切りの横線を選択すると、その選択が無効であることを示すメッセージが返されます。

次に、展開された選択検査 Widget の例と、さまざまな状態の折りたたまれた選択検査 Widget の例を示します。



A. フォーカスがあるときの選択検査 Widget B. 選択 Widget、有効状態 C. 選択 Widget、必須状態 D. 選択 Widget、無効状態

選択検査 Widget には、有効、無効、必須などのいくつかの状態が含まれます。これらの状態のプロパティは、使用する検査結果に応じて、プロパティインスペクタを使用して変更できます。検査が実行されるタイミングは、さまざまに設定できます。たとえば、ユーザーが Widget の外側をクリックしたとき、選択したとき、フォームを送信しようとしたときなどに設定できます。

**初期状態** ページがブラウザに読み込まれたとき、またはユーザーがフォームをリセットしたとき。

**フォーカス状態** ユーザーが Widget をクリックしたとき。

**有効状態** ユーザーが有効な項目を選択し、フォームを送信できる状態になったとき。

**無効状態** ユーザーが無効な項目を選択したとき。

**必須状態** ユーザーが有効な項目を選択していないとき。

ユーザーとのインタラクションの中で、選択検査 Widget によってこれらのいずれかの状態が入力されると、実行時に Spry フレームワークロジックにより、この Widget の HTML コンテナに特定の CSS クラスが適用されます。たとえば、ユーザーがフォームを送信しようとしてメニューから項目が選択されていなかった場合に、「項目を選択してください。」というエラーメッセージが表示されるように Widget にクラスが適用されます。エラーメッセージのスタイルと表示状態を制御するルールは、この Widget に対応する "SpryValidationSelect.css" ファイルに存在します。

選択検査 Widget のデフォルト HTML コードは、通常はフォームの内部に配置され、コンテナ span タグで構成されます。このタグはテキスト領域の select タグを囲みます。選択検査 Widget の HTML コードには、ドキュメントのヘッド内と Widget の HTML コードの後に script タグも含まれます。

選択検査 Widget の HTML コードには、ドキュメントのヘッド内と Widget の HTML コードの後に script タグも含まれます。ドキュメントのヘッド内の script タグは、選択 Widget に関連するすべての JavaScript 関数を定義します。Widget コードの後の script タグは、Widget をインタラクティブにする JavaScript オブジェクトを作成します。

選択検査 Widget の HTML コードを次に示します。

```
<head>
...
<!-- Link the Spry Validation Select JavaScript library -->
<script src="SpryAssets/SpryValidationSelect.js" type="text/javascript"></script>
<!-- Link the CSS style sheet that styles the widget -->
<link href="SpryAssets/SpryValidationSelect.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" name="form1" method="post" action="">
    <!-- Create the select widget and assign a unique id-->
    <span id="spryselect1">
      <select name="select1" id="select1">
        <!-- Add items and values to the widget-->
        <option>--Please select an item--</option>
        <option value="item1">Item 1</option>
        <option value="item2">Item 2</option>
        <option value="-1">Invalid Item</option>
        <option value="item3">Item 3</option>
        <option>Empty Item</option>
      </select>
      <!--Add an error message-->
      <span class="selectRequiredMsg">Please select an item.</span>
    </span>
  </form>
  <!-- Initialize the Validation Select widget object-->
  <script type="text/javascript">
var spryselect1 = new Spry.Widget.ValidationSelect("spryselect1");
  </script>
</body>
```

このコードでは、new JavaScript 演算子によって、選択 Widget オブジェクトが初期化され、span コンテンツが spryselect1 の ID を使用して静的 HTML コードからインタラクティブなページエレメントに変換されます。

Widget 内のエラーメッセージの span タグでは、CSS クラスが適用されています。このクラス (デフォルトでは display:none; に設定されています) は、エラーメッセージのスタイルと表示状態を制御し、対応する "SpryValidationSelect.css" ファイルに存在します。Widget がユーザーとのインタラクションの結果としてさまざまな状態を入力した場合、Spry は Widget のコンテナにさまざまなクラスを配置し、それによってエラーメッセージクラスに影響を与えます。

その他のエラーメッセージを選択検査 Widget に追加するには、エラーメッセージのテキストを保持する span タグ (またはその他の種類のタグ) を作成します。次に、CSS クラスを適用することにより、Widget の状態に応じて、メッセージの表示、非表示を切り替えることができます。

デフォルトの選択検査 Widget の状態の外観は、"SpryValidationSelect.css" ファイル内の対応する CSS ルールを編集することで変更できます。たとえば、状態の背景色を変更するには、スタイルシートで対応するルールを編集するか (まだルールが存在しない場合は) 新しいルールを追加します。

#### 選択 Widget 構造に使用されるさまざまなタグ

上の例では、span タグによって Widget の構造が作成されます。

```
Container SPAN
  SELECT tag
  Error message SPAN
```

ただし、Widget は、ほとんどすべてのコンテナタグを使用して作成できます。

```
Container DIV
  SELECT tag
  Error Message P
```

Spry では (タグ自体ではなく) タグ ID を使用して Widget が作成されます。Spry では、エラーメッセージも、エラーメッセージを含めるために使用される実際のタグとは関係のない CSS コードを使用して表示されます。

Widget コンストラクタに渡された ID は特定の HTML エlement を識別します。コンストラクタはこの Element を検出し、識別されたコンテナ内で対応する `select` タグを見つけます。コンストラクタに渡された ID が ( コンテナタグではなく ) `select` タグの ID である場合、コンストラクタは検査トリガを直接 `select` タグに添付します。ただし、コンテナタグが存在しない場合は、Widget はエラーメッセージを表示できず、さまざまな検査状態によって `select` タグ Element の外観 ( 背景色など ) だけが変更されます。

**注意:** 複数の `select` タグは、同じ HTML Widget コンテナの内側では動作しません。各選択リストは、独自の Widget である必要があります。

## 選択検査 Widget の CSS コード

"SpryValidationSelect.css" ファイルには、選択検査 Widget とそのエラーメッセージのスタイルを設定するルールが含まれます。このルールを編集して、Widget およびエラーメッセージの外観と印象に関するスタイルを設定できます。CSS ファイル内のルールの名前は、Widget の HTML コードで指定されたクラスの名前に対応します。

"SpryValidationSelect.css" ファイルの CSS コードを次に示します。

```
/*Validation Select styling classes*/
.selectRequiredMsg, .selectInvalidMsg {
    display: none;
}
.selectRequiredState .selectRequiredMsg,
.selectInvalidState .selectInvalidMsg {
    display: inline;
    color: #CC3333;
    border: 1px solid #CC3333;
}
.selectValidState select, select.selectValidState {
    background-color: #B8F5B1;
}
select.selectRequiredState, .selectRequiredState select,
select.selectInvalidState, .selectInvalidState select {
    background-color: #FF9F9F;
}
.selectFocusState select, select.selectFocusState {
    background-color: #FFFCC;
}
```

"SpryValidationSelect.css" ファイルには、特定のルールに対するコードと目的を説明するさまざまなコメントも含まれます。詳細については、ファイル内のコメントを参照してください。

## 選択検査 Widget の挿入

**1** "SpryValidationSelect.js" ファイルを検索して、Web サイトに追加します。"SpryValidationSelect.js" ファイルは、Adobe Labs からダウンロードした Spry ディレクトリにある `widgets/selectvalidation` ディレクトリ内で検索できます。詳細については、3 ページの「ファイルの準備」を参照してください。

たとえば、Web サイトのルートフォルダ内に **SpryAssets** という名前のフォルダを作成し、このフォルダに "SpryValidationSelect.js" ファイルをアップロードします。"SpryValidationSelect.js" ファイルには、選択 Widget をインタラクティブにするために必要な情報がすべて含まれます。

**2** "SpryValidationSelect.css" ファイルを検索して、Web サイトに追加します。このファイルの追加先として、メインディレクトリである SpryAssets ディレクトリ、または CSS ディレクトリがある場合はそのディレクトリを選択できます。

**3** 選択 Widget の追加先となる Web ページを開き、このページのヘッドタグに次の `script` タグを挿入してこのページに "SpryValidationSelect.js" ファイルをリンクします。

```
<script src="SpryAssets/SpryValidationSelect.js" type="text/javascript"></script>
```

"SpryValidationSelect.js" ファイルへのファイルパスが正しいことを確認します。このパスは、Web サイト内のどこにファイルを配置したかによって異なります。

**4** ページのヘッドタグに次の `link` タグを挿入して、Web ページに "SpryValidationSelect.css" ファイルをリンクします。

```
<link href="SpryAssets/SpryValidationSelect.css" rel="stylesheet" type="text/css" />
```

"SpryValidationSelect.css" ファイルへのファイルパスが正しいことを確認します。このパスは、Web サイト内のどこにファイルを配置したかによって異なります。

**5** 選択リストをページに追加し、名前と一意の ID を指定します。

```
<select name="myselect" id="myselect"></select>
```

**6** 次のように、選択リストにオプションを追加します。

```
<select name="myselect" id="myselect">
  <option>--Please select an item--</option>
  <option value="item1">Item 1</option>
  <option value="item2">Item 2</option>
  <option value="-1">Invalid Item</option>
  <option value="item3">Item 3</option>
  <option>Empty Item</option>
</select>
```

**7** 選択リストの周囲にコンテナを追加するには、span タグを select タグの周囲に挿入し、次のように一意の ID を Widget に割り当てます。

```
<span id="spryselect1">
  <select name="myselect" id="myselect">
    <option>--Please select an item--</option>
    <option value="item1">Item 1</option>
    <option value="item2">Item 2</option>
    <option value="-1">Invalid Item</option>
    <option value="item3">Item 3</option>
    <option>Empty Item</option>
  </select>
</span>
```

**8** Spry 選択検査オブジェクトを初期化するには、次の script ブロックを Widget に対応する HTML コードの後に挿入します。

```
<script type="text/javascript">
  var spryselect1 = new Spry.Widget.ValidationSelect("spryselect1");
</script>
```

new JavaScript 演算子によって、選択 Widget オブジェクトが初期化され、span タグコンテンツが spryselect1 の ID を使用して静的 HTML コードからインタラクティブな選択オブジェクトに変換されます。Spry.Widget.ValidationSelect メソッドは、選択オブジェクトを作成する Spry フレームワークのコンストラクタです。このオブジェクトを初期化するために必要な情報は、この手順の開始時にリンクした SpryValidationSelect.js JavaScript ライブラリに含まれています。

選択リストのコンテナ span タグの ID が Spry.Widgets.ValidationSelect メソッドで指定した ID パラメータと一致することを確認します。JavaScript の呼び出しが Widget に対応する HTML コードの後に行われることを確認します。

**9** ページを保存します。

コード全体を次に示します。

```
<head>
...
<script src="SpryAssets/SpryValidationSelect.js" type="text/javascript"></script>
<link href="SpryAssets/SpryValidationSelect.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <span id="spryselect1">
    <select name="myselect" id="myselect">
      <option>--Please select an item--</option>
      <option value="item1">Item 1</option>
      <option value="item2">Item 2</option>
      <option value="-1">Invalid Item</option>
      <option value="item3">Item 3</option>
      <option>Empty Item</option>
    </select>
  </span>
<script type="text/javascript">
  var spryselect1 = new Spry.Widget.ValidationSelect("spryselect1");
</script>
</body>
```

## エラーメッセージの表示

❖ エラーメッセージを表示するための `span` タグ (またはその他の種類のタグ) を作成し、次のように適切なクラスを割り当てます。

```
<span id="spryselect1">
  <select name="select1" id="select1">
    <option>--Please select an item--</option>
    <option value="item1">Item 1</option>
    . . .
  </select>
  <span class="selectRequiredMsg">Please select an item.</span>
</span>
```

`selectRequiredMsg` ルールは `"SpryValidationSelect.css"` ファイル内に配置され、デフォルトでは `display:none` に設定されています。ユーザーとのインタラクションの中で `Widget` によって別の状態が入力されると、`Spry` により、この `Widget` のコンテナに適切なクラス (状態クラス) が適用されます。このアクションは、エラーメッセージクラスに影響を与えるので、結果として、エラーメッセージの外観に影響します。

たとえば、`"SpryValidationSelect.css"` ファイルの CSS ルールの一部は、次のようになります。

```
.selectRequiredMsg, .selectInvalidMsg {
  display: none;
}
.selectRequiredState .selectRequiredMsg,
.selectInvalidState .selectInvalidMsg {
  display: inline;
  color: #CC3333;
  border: 1px solid #CC3333;
}
```

デフォルトでは、状態クラスは `Widget` コンテナに適用されていないため、ページがブラウザに読み込まれるときに、上の HTML コード例のエラーメッセージテキストに適用されているのは `.selectRequiredMsg` クラスだけです。このルールのプロパティと値のペアは `display:none` なので、メッセージは非表示のままになります。ただし、ユーザーが選択を行わない場合は、次のように `Spry` によって `Widget` コンテナに適切なクラスが適用されます。

```
<span id="spryselect1" class="selectRequiredState">
  <select name="select1" id="select1">
    <option>--Please select an item--</option>
    <option value="item1">Item 1</option>
    . . .
  </select>
  <span class="selectRequiredMsg">Please select an item.</span>
</span>
```

上の CSS コードでは、コンテキストセレクタである `.selectRequiredState .selectRequiredMsg` を持つ状態ルールによって、エラーメッセージテキストを非表示にするデフォルトのエラーメッセージルールが上書きされます。このため、Spry によって状態クラスが Widget コンテナに適用されると、状態ルールによって Widget の外観が決定され、エラーメッセージは 1 ピクセルの実線のボーダーに囲まれて赤でインラインで表示されます。

デフォルトのエラーメッセージクラスとその説明のリストを次に示します。これらのクラスを変更して、名前を変更することができます。その場合は、必ずコンテキストセレクタも変更してください。

| エラーメッセージクラス                     | 説明                                  |
|---------------------------------|-------------------------------------|
| <code>.selectRequiredMsg</code> | Widget が必須状態を入力すると、エラーメッセージを表示します。  |
| <code>.selectInvalidMsg</code>  | Widget が無効な状態を入力すると、エラーメッセージを表示します。 |

**注意：**状態関連のクラスの名前は、Spry フレームワークの一部としてハードコーディングされているため、変更することはできません。

### 検査を実行するタイミングの指定

デフォルトでは、選択検査 Widget は、ユーザーが送信ボタンをクリックしたときに検査します。ただし、その他に、`blur` または `change` の 2 つのオプションを設定することもできます。`validateOn:["blur"]` パラメータを指定すると、Widget は、ユーザーが選択リストの外側をクリックしたときに常に検査します。`validateOn:["change"]` パラメータを指定すると、Widget は、ユーザーが選択を行ったときに検査します。

❖ 検査を実行するタイミングを指定するには、次のように、`validateOn` パラメータをコンストラクタに追加します。

```
<script type="text/javascript">
  var spryselect1 = new Spry.Widget.ValidationSelect("spryselect1", {validateOn:["blur"]});
</script>
```

`validateOn` パラメータに値を 1 つしか指定しない場合は、簡単にするため、角カッコを省略できます (たとえば、`validateOn:"blur"`)。ただし、両方のパラメータを指定する場合は (`validateOn:["blur","change"]`)、シンタックスに角カッコが必要です。

### 選択リストの必須状態の変更

デフォルトでは、選択検査 Widget ではユーザーがフォームを送信する前に選択を行う必要があります。ただし、ユーザーに対して選択をオプションにすることもできます。

❖ 選択リストの必須状態を変更するには、次のように `isRequired` プロパティをコンストラクタに追加し、その値を `false` に設定します。

```
<script type="text/javascript">
  var selectwidget1 = new Spry.Widget.ValidationSelect("selectwidget1", {isRequired:false});
</script>
```

### 無効な値の指定

関連付けられたメニュー項目をユーザーが選択すると無効とされる値を指定できます。たとえば、無効な値として `-1` を指定し、その値をオプションタグに割り当てると、ユーザーがメニュー項目を選択したときに Widget はエラーメッセージを返します。

1 次のように、オプションタグに負の値を割り当てます。

```
<option value="-1"> ----- </option>
```

2 次のように、無効なオプションと指定した値を Widget コンストラクタに追加します。

```
<script type="text/javascript">
  var selectwidget1 = new Spry.Widget.ValidationSelect("selectwidget1", {invalidValue:"-1"});
</script>
```

## 選択検査 Widget のカスタマイズ

"SpryValidationSelect.css" ファイルには、選択検査 Widget のデフォルトのスタイル設定が用意されています。ただし、該当する CSS ルールを変更することによって Widget をカスタマイズできます。"SpryValidationSelect.css" ファイルの CSS ルールでは、Widget の HTML コードの関連するエレメントと同じクラス名が使用されるため、Widget とそのエラー状態に対応する CSS ルールを簡単に把握できます。

"SpryValidationSelect.css" ファイルは、カスタマイズを開始する前に Web サイトに配置しておく必要があります。詳細については、3 ページの「ファイルの準備」を参照してください。

### 選択検査 Widget のスタイル設定 (一般的な方法)

- 1 "SpryValidationSelect.css" ファイルを開きます。
- 2 Widget の変更する部分の CSS ルールを見つけます。たとえば、選択 Widget の必須状態の背景色を変更するには、"SpryValidationSelect.css" ファイルの .selectRequiredState ルールを編集します。
- 3 CSS ルールを変更してファイルを保存します。

"SpryValidationSelect.css" ファイルには、特定のルールに対するコードと目的を説明するさまざまなコメントが含まれます。詳細については、ファイル内のコメントを参照してください。

### 選択検査 Widget のエラーメッセージのテキストのスタイル設定

選択検査 Widget のエラーメッセージは、デフォルトでは、1 ピクセルの実線のボーダーで囲まれた赤いテキストで表示されます。

❖ 選択検査 Widget のエラーメッセージのテキストスタイルを変更するには、次の表を使用して適切な CSS ルールを見つけ、デフォルトのプロパティを変更するか、または独自のテキストスタイルプロパティおよび値を追加します。

| スタイルを設定するテキスト | 関連する CSS ルール  | 変更する関連プロパティ                                |
|---------------|---|--|
| エラーメッセージのテキスト | .selectRequiredState .selectRequiredMsg,<br>.selectInvalidState .selectInvalidMsg | color: #CC3333; border: 1px solid #CC3333; |

### 選択検査 Widget の背景色の変更

❖ さまざまな状態の選択検査 Widget の背景色を変更するには、次の表を使用して適切な CSS ルールを見つけ、デフォルトの背景色値を変更します。

| 変更する背景色              | 関連する CSS ルール  | 変更する関連プロパティ                |
|----------------------|---|----------------------------|
| 有効状態の Widget の背景色    | .selectValidState select,<br>select.selectValidState  | background-color: #B8F5B1; |
| 無効状態の Widget の背景色    | select.selectRequiredState,<br>.selectRequiredState select,<br>select.selectInvalidState,<br>.selectInvalidState select | background-color: #FF9F9F; |
| フォーカスがある Widget の背景色 | .selectFocusState select,<br>select.selectFocusState  | background-color: #FFFFCC; |

### 状態関連のクラス名のカスタマイズ

CSS 内のルールと HTML コード内のクラス名を変更することによって、エラーメッセージ関連のクラス名を独自のクラス名に置き換えることはできますが、ビヘイビアは Spry フレームワークの一部としてハードコーディングされているため、状態関連のクラス名を変更または置換することはできません。ただし、Widget コンストラクタの 3 番目のパラメータで新しい値を指定することによって、デフォルトの状態関連のクラス名を独自の名前で上書きすることができます。

❖ Widget の状態関連のクラス名を変更するには、次のように、上書きを行うオプションのいずれかを Widget コンストラクタの 3 番目のパラメータに追加し、カスタムクラス名を指定します。

```
<script type="text/javascript">
  var spryselect1 = new Spry.Widget.ValidationSelect("spryselect1", {requiredClass:"required"});
</script>
```

次の表に、組み込みの状態関連のクラス名を上書きするために使用できるオプションのリストを示します。

| オプション         | 説明                                  |
|---------------|-------------------------------------|
| requiredClass | "selectRequiredState" 組み込み値を上書きします。 |
| validClass    | "selectValidState" 組み込み値を上書きします。    |
| focusClass    | "selectFocusState" 組み込み値を上書きします。    |
| invalidClass  | "selectInvalidState" 組み込み値を上書きします。  |

## チェックボックス検査 Widget の操作

### チェックボックス検査 Widget の概要と構造

Spry チェックボックス検査 Widget は、ユーザーが HTML コードフォーム内のあるチェックボックスを選択したとき、または選択しなかったときに、有効状態または無効状態を表示する（単独またはグループの）チェックボックスです。たとえば、ユーザーが 3 つのオプションを選択する必要があるフォームにチェックボックス検査 Widget を追加します。ユーザーが 3 つのオプションを選択していない場合は、最小選択数が満たされなかったことを示すメッセージが返されます。

さまざまな状態のチェックボックス検査 Widget の例を次に示します。

A

|   |                                     |                                     |
|---|-------------------------------------|-------------------------------------|
| <input checked="" type="checkbox"/> Entertainment | <input type="checkbox"/> Computers  | <input type="checkbox"/> Sports     |
| <input type="checkbox"/> Health                   | <input type="checkbox"/> Finance    | <input type="checkbox"/> Travel     |
| <input type="checkbox"/> Music                    | <input type="checkbox"/> Technology | <input type="checkbox"/> Publishing |

選択範囲の最小数に達していません。

Submit

B

Check me!

項目を選択してください。

Submit

A. チェックボックス検査 Widget グループ、最小選択数状態 B. チェックボックス検査 Widget、必須状態

チェックボックス検査 Widget には、有効、無効、必須などのいくつかの状態が含まれます。これらの状態のプロパティは、使用する検査結果に応じて、プロパティインスペクタを使用して変更できます。検査が実行されるタイミングは、さまざまに設定できます。たとえば、ユーザーが Widget の外側をクリックしたとき、選択したとき、フォームを送信しようとしたときなどに設定できます。

**初期状態** ページがブラウザに読み込まれたとき、またはユーザーがフォームをリセットしたとき。

**有効状態** ユーザーが適切な数のオプションを選択し、フォームを送信できるようになったとき。

**必須状態** ユーザーが必要な選択を行わなかったとき。

**最小選択数状態** ユーザーが選択したチェックボックスの数が必要最小数に満たないとき。

**最大選択数状態** ユーザーが選択したチェックボックスの数が許容最大数を超えているとき。

ユーザーとのインタラクションの中で、チェックボックス検査 Widget によってこれらのいずれかの状態が入力されると、実行時に Spry フレームワークロジックにより、この Widget の HTML コンテナに特定の CSS クラスが適用されます。たとえば、ユーザーがフォームをオプションを選択せずに送信しようとする、と、「項目を選択してください。」というエラーメッセージを表示するクラスが Widget に適用されます。エラーメッセージのスタイルと表示状態を制御するルールは、この Widget に対応する "SpryValidationCheckbox.css" ファイルに存在します。

チェックボックス検査 Widget のデフォルト HTML コードは、通常はフォームの内部に配置され、コンテナ span タグで構成されます。このタグはチェックボックスの input type="checkbox" タグを囲みます。チェックボックス検査 Widget の HTML コードには、ドキュメントのヘッド内と Widget の HTML コードの後に script タグも含まれます。

チェックボックス検査 Widget の HTML コードには、ドキュメントのヘッド内と Widget の HTML コードの後に script タグも含まれます。ドキュメントのヘッド内の script タグは、チェックボックス Widget に関連するすべての JavaScript 関数を定義します。Widget コードの後の script タグは、チェックボックスをインタラクティブにする JavaScript オブジェクトを作成します。

チェックボックス検査 Widget の HTML コードを次に示します。

```
<head>
...
<!-- Link the Spry Validation Checkbox JavaScript library -->
<script src="SpryAssets/SpryValidationCheckbox.js" type="text/javascript"></script>
<!-- Link the CSS style sheet that styles the widget -->
<link href="SpryAssets/SpryValidationCheckbox.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form id="form1" name="form1" method="post" action="">
    <!-- Create the checkbox widget and assign a unique id-->
    <span id="sprycheckbox1">
      <input type="checkbox" name="checkbox1" value="1"/>
      <input type="checkbox" name="checkbox2" value="2"/>
      <!--Add an error message-->
      <span class="CheckboxRequiredMsg">Please make a selection.</span>
    </span>
  </form>
  <!-- Initialize the Validation Checkbox widget object-->
  <script type="text/javascript">
    var sprycheckbox1 = new Spry.Widget.ValidationCheckbox("sprycheckbox1");
  </script>
</body>
```

このコードでは、new JavaScript 演算子によって、チェックボックス Widget オブジェクトが初期化され、span コンテンツが sprycheckbox1 の ID を使用して静的 HTML コードからインタラクティブなページエレメントに変換されます。

Widget 内のエラーメッセージの span タグでは、CSS クラスが適用されています。このクラス (デフォルトでは display:none; に設定されています) は、エラーメッセージのスタイルと表示状態を制御し、対応する "SpryValidationCheckbox.css" ファイルに存在します。Widget がユーザーとのインタラクションの結果としてさまざまな状態を入力した場合、Spry は Widget のコンテナにさまざまなクラスを配置し、それによってエラーメッセージクラスに影響を与えます。

その他のエラーメッセージをチェックボックス検査 Widget に追加するには、エラーメッセージのテキストを保持する span タグ (またはその他の種類のタグ) を作成します。次に、CSS クラスを適用することにより、Widget の状態に応じて、メッセージの表示、非表示を切り替えることができます。

デフォルトのチェックボックス検査 Widget の状態の外観は、"SpryValidationCheckbox.css" ファイル内の対応する CSS ルールを編集することで変更できます。たとえば、状態の背景色を変更するには、スタイルシートで対応するルールを編集するか (まだルールが存在しない場合は) 新しいルールを追加します。

#### チェックボックス検査 Widget 構造に使用されるさまざまなタグ

上の例では、span タグによって Widget の構造が作成されます。

```
Container SPAN
  INPUT type="checkbox"
  Error message SPAN
```

ただし、Widget は、ほとんどすべてのコンテナタグを使用して作成できます。

```
Container DIV
  INPUT type="checkbox"
  Error Message P
```

Spry では ( タグ自体ではなく ) タグ ID を使用して Widget が作成されます。Spry では、エラーメッセージも、エラーメッセージを含めるために使用される実際のタグとは関係のない CSS コードを使用して表示されます。

Widget コンストラクタに渡された ID は特定の HTML エlement を識別します。コンストラクタはこの Element を検出し、識別されたコンテナ内で対応する input タグを見つけます。コンストラクタに渡された ID が ( コンテナタグではなく ) input タグの ID である場合、コンストラクタは検査トリガを直接 input タグに添付します。ただし、コンテナタグが存在しない場合は、Widget はエラーメッセージを表示できず、さまざまな検査状態によって input タグ Element の外観 ( 背景色など ) だけが変更されます。

## チェックボックス検査 Widget の CSS コード

"SpryValidationCheckbox.css" ファイルには、チェックボックス検査 Widget とそのエラーメッセージのスタイルを設定するルールが含まれます。このルールを編集して、Widget およびエラーメッセージの外観と印象に関するスタイルを設定できます。CSS ファイル内のルールの名前は、Widget の HTML コードで指定されたクラスの名前に対応します。

"SpryValidationCheckbox.css" ファイルの CSS コードを次に示します。

```
/*Validation Checkbox styling classes*/
.checkboxRequiredMsg, .checkboxMinSelectionsMsg, .checkboxMaxSelectionsMsg{
    display: none;
}
.checkboxRequiredState .checkboxRequiredMsg,
.checkboxMinSelectionsState .checkboxMinSelectionsMsg,
.checkboxMaxSelectionsState .checkboxMaxSelectionsMsg {
    display: inline;
    color: #CC3333;
    border: 1px solid #CC3333;
}
```

"SpryValidationCheckbox.css" ファイルには、特定のルールに対するコードと目的を説明するさまざまなコメントも含まれます。詳細については、ファイル内のコメントを参照してください。

## チェックボックス検査 Widget の挿入

1 "SpryValidationCheckbox.js" ファイルを検索して、Web サイトに追加します。"SpryValidationCheckbox.js" ファイルは、Adobe Labs からダウンロードした Spry ディレクトリにある widgets/checkboxvalidation ディレクトリ内で検索できます。詳細については、3 ページの「ファイルの準備」を参照してください。

たとえば、Web サイトのルートフォルダ内に **SpryAssets** という名前のフォルダを作成し、このフォルダに "SpryValidationCheckbox.js" ファイルをアップロードします。"SpryValidationCheckbox.js" ファイルには、チェックボックス Widget をインタラクティブにするために必要な情報がすべて含まれます。

2 "SpryValidationCheckbox.css" ファイルを検索して、Web サイトに追加します。このファイルの追加先として、メインディレクトリである SpryAssets ディレクトリ、または CSS ディレクトリがある場合はそのディレクトリを選択できます。

3 チェックボックス Widget の追加先となる Web ページを開き、このページのヘッドタグに次の script タグを挿入してこのページに "SpryValidationCheckbox.js" ファイルをリンクします。

```
<script src="SpryAssets/SpryValidationCheckbox.js" type="text/javascript"></script>
```

"SpryValidationCheckbox.js" ファイルへのファイルパスが正しいことを確認します。このパスは、Web サイト内のどこにファイルを配置したかによって異なります。

4 ページのヘッドタグに次の link タグを挿入して、Web ページに "SpryValidationCheckbox.css" ファイルをリンクします。

```
<link href="SpryAssets/SpryValidationCheckbox.css" rel="stylesheet" type="text/css" />
```

"SpryValidationCheckbox.css" ファイルへのファイルパスが正しいことを確認します。このパスは、Web サイト内のどこにファイルを配置したかによって異なります。

**5** チェックボックスをページに追加し、それらの名前と値を割り当てます。追加できるチェックボックスの数は制限されていません。

```
<input type="checkbox" name="checkbox1" value="1"/>
<input type="checkbox" name="checkbox2" value="2"/>
```

**6** 次のように span タグを input タグの周囲に挿入し、一意の ID を Widget に割り当てることによって、チェックボックスの周囲にコンテナを追加します。

```
<span id="sprycheckbox1">
  <input type="checkbox" name="checkbox1" value="1"/>
  <input type="checkbox" name="checkbox2" value="2"/>
</span>
```

**7** Spry チェックボックス検査オブジェクトを初期化するには、次の script ブロックを Widget に対応する HTML コードの後に挿入します。

```
<script type="text/javascript">
  var sprycheckbox1 = new Spry.Widget.ValidationCheckbox("sprycheckbox1");
</script>
```

new JavaScript 演算子によって、チェックボックス Widget オブジェクトが初期化され、span タグコンテンツが sprycheckbox1 の ID を使用して静的 HTML コードからインタラクティブなチェックボックスオブジェクトに変換されます。Spry.Widget.ValidationCheckbox メソッドは、チェックボックスオブジェクトを作成する Spry フレームワークのコンストラクタです。このオブジェクトを初期化するために必要な情報は、この手順の開始時にリンクした SpryValidationCheckbox.js JavaScript ライブラリに含まれています。

チェックボックス Widget のコンテナの span タグの ID が Spry.Widgets.ValidationCheckbox メソッドで指定した ID パラメータと一致することを確認します。JavaScript の呼び出しが Widget に対応する HTML コードの後に行われることを確認します。

**8** ページを保存します。

コード全体を次に示します。

```
<head>
...
<script src="SpryAssets/SpryValidationCheckbox.js" type="text/javascript"></script>
<link href="SpryAssets/SpryValidationCheckbox.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <span id="sprycheckbox1">
    <input type="checkbox" name="checkbox1" value="1"/>
    <input type="checkbox" name="checkbox2" value="2"/>
  </span>
  <script type="text/javascript">
    var sprycheckbox1 = new Spry.Widget.ValidationCheckbox("sprycheckbox1");
  </script>
</body>
```

## エラーメッセージの表示

❖ エラーメッセージを表示するための span タグ (またはその他の種類のタグ) を作成し、次のように適切なクラスを割り当てます。

```
<span id="sprycheckbox1">
  <input type="checkbox" name="checkbox1" value="1"/>
  <input type="checkbox" name="checkbox2" value="2"/>
  <span class="checkboxRequiredMsg">Please make a selection.</span>
</span>
```

checkboxRequiredMsg ルールは SpryValidationCheckbox.css ファイル内に配置され、デフォルトでは display:none に設定されています。ユーザーとのインタラクションの中で Widget によって別の状態が入力されると、Spry により、この Widget のコンテナに適切なクラス (状態クラス) が適用されます。このアクションは、エラーメッセージクラスに影響を与えるので、結果として、エラーメッセージの外観に影響します。

たとえば、"SpryValidationCheckbox.css" ファイルの CSS ルールは、次のようになります。

```
.checkboxRequiredMsg, .checkboxMinSelectionsMsg, .checkboxMaxSelectionsMsg{
  display: none;
}
.checkboxRequiredState .checkboxRequiredMsg,
.checkboxMinSelectionsState .checkboxMinSelectionsMsg,
.checkboxMaxSelectionsState .checkboxMaxSelectionsMsg {
  display: inline;
  color: #CC3333;
  border: 1px solid #CC3333;
}
```

デフォルトでは、状態クラスは Widget コンテナに適用されていないため、ページがブラウザに読み込まれるときに、上の HTML コード例のエラーメッセージテキストに適用されているのは checkboxRequiredMsg クラスだけです。このルールのプロパティと値のペアは display:none なので、メッセージは非表示のままになります。ただし、ユーザーが選択を行わない場合は、次のように Spry によって Widget コンテナに適切なクラスが適用されます。

```
<span id="sprycheckbox1" class="checkboxRequiredState">
  <input type="checkbox" name="checkbox1" value="1"/>
  <input type="checkbox" name="checkbox2" value="2"/>
  <span class="checkboxRequiredMsg">Please make a selection.</span>
</span>
```

上の CSS コードでは、コンテキストセレクタである .checkboxRequiredState .checkboxRequiredMsg を持つ状態ルールによって、エラーメッセージテキストを非表示にするデフォルトのエラーメッセージルールが上書きされます。このため、Spry によって状態クラスが Widget コンテナに適用されると、状態ルールによって Widget の外観が決定され、エラーメッセージは 1 ピクセルの実線のボーダーに囲まれて赤でインラインで表示されます。

デフォルトのエラーメッセージクラスとその説明のリストを次に示します。これらのクラスを変更して、名前を変更することができます。その場合は、必ずコンテキストセレクタも変更してください。

| エラーメッセージクラス               | 説明                                    |
|---------------------------|---------------------------------------|
| .checkboxRequiredMsg      | Widget が必須状態を入力すると、エラーメッセージを表示します。    |
| .checkboxMinSelectionsMsg | Widget が最小選択数状態を入力すると、エラーメッセージを表示します。 |
| .checkboxMaxSelectionsMsg | Widget が最大選択数状態を入力すると、エラーメッセージを表示します。 |

**注意：**状態関連のクラスの名前は、Spry フレームワークの一部としてハードコーディングされているため、変更することはできません。

### 検査を実行するタイミングの指定

デフォルトでは、チェックボックス検査 Widget は、ユーザーが送信ボタンをクリックしたときに検査します。ただし、その他に、blur または change の 2 つのオプションを設定することもできます。validateOn:["blur"] パラメータを指定すると、Widget は、ユーザーが Widget の外側をクリックしたときに常に検査します。validateOn:["change"] パラメータを指定すると、Widget は、ユーザーが選択を行ったときに検査します。

❖ 検査を実行するタイミングを指定するには、次のように、validateOn パラメータをコンストラクタに追加します。

```
<script type="text/javascript">
  var sprycheckbox1 = new Spry.Widget.ValidationCheckbox("sprycheckbox1", {validateOn:["blur"]});
</script>
```

validateOn パラメータに値を 1 つしか指定しない場合は、簡単にするため、角カッコを省略できます (たとえば、validateOn: "blur")。ただし、両方のパラメータを指定する場合は (validateOn: ["blur", "change"])、シンタックスに角カッコが必要です。

### 最小および最大選択オプション数の指定

デフォルトでは、チェックボックス検査 Widget が required になるように設定されます。ただし、ページに複数のチェックボックスを挿入する場合は、選択オプション数の範囲を指定できます。たとえば、1 つのチェックボックス検査 Widget の span タグ内に 6 つのチェックボックスがあり、ユーザーに 3 つ以上のチェックボックスを選択させたい場合は、Widget 全体にそのような要件を設定できます。

❖ 最小選択数または最大選択数を指定するには、次のように、コンストラクタに minSelections プロパティまたは maxSelections プロパティ (あるいはその両方) と値を追加します。

```
<script type="text/javascript">
    var checkboxwidget1 = new Spry.Widget.ValidationCheckbox("checkboxwidget1", {minSelections:value,
maxSelections:value});
</script>
```

### チェックボックスの必須状態の変更

デフォルトでは、チェックボックス検査 Widget ではユーザーがフォームを送信する前に少なくとも 1 つの選択を行う必要があります。ただし、ユーザーに対して選択をオプションにすることもできます。

❖ チェックボックスの必須状態を変更するには、次のように isRequired プロパティをコンストラクタに追加し、その値を false に設定します。

```
<script type="text/javascript">
    var checkboxwidget1 = new Spry.Widget.ValidationCheckbox("checkboxwidget1", {isRequired:false});
</script>
```

### チェックボックス検査 Widget のカスタマイズ

"SpryValidationCheckbox.css" ファイルには、チェックボックス検査 Widget のデフォルトのスタイル設定が用意されています。該当する CSS ルールを変更することによって Widget をカスタマイズできます。

"SpryValidationCheckbox.css" ファイルの CSS ルールでは、Widget の HTML コードの関連するエレメントと同じクラス名が使用されるため、Widget とそのエラー状態に対応する CSS ルールを簡単に把握できます。

"SpryValidationCheckbox.css" ファイルは、カスタマイズを開始する前に Web サイトに配置しておく必要があります。詳細については、3 ページの「ファイルの準備」を参照してください。

#### チェックボックス検査 Widget のスタイル設定 (一般的な方法)

- 1 "SpryValidationCheckbox.css" ファイルを開きます。
- 2 Widget の変更する部分の CSS ルールを見つけます。たとえば、チェックボックス Widget の必須状態の背景色を変更するには、"SpryValidationCheckbox.css" ファイルの .checkboxRequiredState ルールを編集します。
- 3 CSS ルールを変更してファイルを保存します。

"SpryValidationCheckbox.css" ファイルには、特定のルールに対するコードと目的を説明するさまざまなコメントが含まれます。詳細については、ファイル内のコメントを参照してください。

#### チェックボックス検査 Widget のエラーメッセージのテキストのスタイル設定

チェックボックス検査 Widget のエラーメッセージは、デフォルトでは、1 ピクセルの実線のパターンで囲まれた赤いテキストで表示されます。

❖ チェックボックス検査 Widget のエラーメッセージのテキストスタイルを変更するには、次の表を使用して適切な CSS ルールを見つけ、デフォルトのプロパティを変更するか、または独自のテキストスタイルプロパティおよび値を追加します。

| スタイルを設定するテキスト | 関連する CSS ルール   | 変更する関連プロパティ                                |
|---------------|--|--|
| エラーメッセージのテキスト | .checkboxRequiredState<br>.checkboxRequiredMsg,<br>.checkboxMinSelectionsState<br>.checkboxMinSelectionsMsg,<br>.checkboxMaxSelectionsState<br>.checkboxMaxSelectionsMsg | color: #CC3333; border: 1px solid #CC3333; |

### 状態関連のクラス名のカスタマイズ

CSS 内のルールと HTML コード内のクラス名を変更することによって、エラーメッセージ関連のクラス名を独自のクラス名に置き換えることはできますが、jQuery は Spry フレームワークの一部としてハードコーディングされているため、状態関連のクラス名を変更または置換することはできません。ただし、Widget コンストラクタの 3 番目のパラメータで新しい値を指定することによって、デフォルトの状態関連のクラス名を独自の名前で上書きすることができます。

❖ Widget の状態関連のクラス名を変更するには、次のように、上書きを行うオプションのいずれかを Widget コンストラクタの 3 番目のパラメータに追加し、カスタムクラス名を指定します。

```
<script type="text/javascript">
    var sprycheckbox1 = new Spry.Widget.ValidationCheckbox("sprycheckbox1", {requiredClass:"required"});
</script>
```

次の表に、組み込みの状態関連のクラス名を上書きするために使用できるオプションのリストを示します。

| オプション              | 説明   |
|--------------------|--|
| requiredClass      | "checkboxRequiredState" 組み込み値を上書きします。      |
| minSelectionsClass | "checkboxMinSelectionsState" 組み込み値を上書きします。 |
| maxSelectionsClass | "checkboxMaxSelectionsState" 組み込み値を上書きします。 |

## 第3章：Spry XML データセットの使用

Spry XML データセットは、XML データソースファイルのデータを Web ページに表示するために使用できる JavaScript オブジェクトです。そのデータを使用して、サイトビジターの選択の変化に応じて更新されるマスター領域や詳細領域をページ上に作成できます。

### Spry XML データセットと動的領域について

#### Spry XML データセットの基本的な概要

Spry データセットは、本質的には JavaScript オブジェクトです。Web ページで少数のコードスニペットを使用するだけで、このオブジェクトを作成して、ユーザーがブラウザでページを開いたときに XML ソースのデータをロードすることができます。データがロードされると、データセットは、行と列を含む標準のテーブルとして表現可能な平面化された配列の XML データになります。

たとえば、次のような情報を含む "cafetownsend.xml" という XML ソースファイルがあったとします。

```
<?xml version="1.0" encoding="UTF-8"?>
<specials>
  <menu_item id="1">
    <item>Summer Salad</item>
    <description>organic butter lettuce with apples, blood oranges, gorgonzola, and raspberry vinaigrette.</description>
    <price>7</price>
  </menu_item>
  <menu_item id="2">
    <item>Thai Noodle Salad</item>
    <description>lightly sauteed in sesame oil with baby bok choy, portobello mushrooms, and scallions.</description>
    <price>8</price>
  </menu_item>
  <menu_item id="3">
    <item>Grilled Pacific Salmon</item>
    <description>served with new potatoes, diced beets, Italian parlsey, and lemon zest.</description>
    <price>16</price>
  </menu_item>
</specials>
```

Web ページで XPath を使用して目的のデータ（この例の場合はこの XML ファイルの specials/menu\_item ノード）を指定すると、XML データが次のテーブルのようなオブジェクトの配列（行）とプロパティ（列）に平面化されます。

| @id | item                   | description   | price |
|-----|------------------------|---|-------|
| 1   | Summer salad           | organic butter lettuce with apples, blood oranges, gorgonzola, and raspberry vinaigrette. | 7     |
| 2   | Thai Noodle Salad      | lightly sauteed in sesame oil with baby bok choy, portobello mushrooms, and scallions.    | 8     |
| 3   | Grilled Pacific Salmon | served with new potatoes, diced beets, Italian parlsey, and lemon zest.                   | 16    |

このデータセットには、各メニュー品目に対応する行と、@id、item、description、および price の各列が含まれています。これらの列は、XML の specials/menu\_item ノードの子ノードと、menu\_item タグまたは menu\_item タグの子タグに含まれている属性を表しています。

データセットにはこのほかに、**ds\_RowID** という組み込みのデータ参照も含まれています (ここには示されていません)。これは、後にデータを表示する際に役に立ちます。そのほか、**ds\_RecordCount** や **ds\_CurrentRow** など、データの表示を操作するために使用できる組み込みのデータ参照も含まれています。

Spry データセットオブジェクトは、`Spry.Data.XMLDataSet` コンストラクタで XPath を使用して作成します。XPath は、データセットのデフォルトの構造を定義します。たとえば、XPath を使用して、3 つの子ノードを含む繰り返し XML ノードを選択した場合、データセットには、各繰り返しノードに対応する行と、3 つの子ノードのそれぞれに対応する列が含まれます。繰り返しノードや子ノードに属性が含まれている場合は、各属性に対応する列も作成されます。

XPath を指定しない場合は、XML ソースのすべてのデータがデータセットに含まれます。

データセットが作成されたら、そのデータセットオブジェクトを使用して簡単にデータを表示および管理できます。たとえば、XML データを表示する簡単なテーブルを作成し、簡単なメソッドとプロパティを使用して、データのリロード、ソート、フィルタ処理、ページ処理などを行うことができます。

次の例では、`dsSpecials` という Spry データセットを作成し、"**cafetownsend.xml**" という XML ファイルのデータをロードしています。

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  <title>Spry Example</title>
  <!--Link the Spry libraries-->
  <script type="text/javascript" src="includes/xpath.js"></script>
  <script type="text/javascript" src="includes/SpryData.js"></script>
  <!--Create a data set object-->
  <script type="text/javascript">
    var dsSpecials = new Spry.Data.XMLDataSet("data/cafetownsend.xml", "specials/menu_item");
  </script>
</head>
. . .
<body>
</body>
```

**注意：**このドキュメントの例は参照のみを目的としており、実行用には作られていません。実際に使用できるサンプルについては、Adobe Labs の "Spry" フォルダにある "demos" フォルダを参照してください。

この例の 1 つ目の script タグでは、オープンソースの XPath ライブラリを、最終的に XML データを表示するページにリンクしています。この XPath ライブラリを使用すると、データセットを作成するときにより複雑な XPath を指定できるようになります。

```
<script type="text/javascript" src="includes/xpath.js"></script>
```

2 つ目の script ブロックでは、Spry データライブラリの "`SpryData.js`" をリンクしています。"`SpryData.js`" は、サーバー上の "**includes**" というフォルダに格納されています。

```
<script type="text/javascript" src="includes/SpryData.js"></script>
```

Spry データライブラリは XPath ライブラリに依存しています。したがって、常に XPath ライブラリを先にリンクすることが重要です。

3 つ目の script ブロックには、`dsSpecials` データセットを作成するステートメントが含まれています。XML ソースファイル "**cafetownsend.xml**" は、サーバー上の "**data**" というフォルダに格納されています。

```
var dsSpecials = new Spry.Data.XMLDataSet("data/cafetownsend.xml", "specials/menu_item");
```

**注意：**JavaScript や XML では大文字と小文字が区別されます。したがって、スクリプトや列の名前では大文字と小文字の区別に注意する必要があります。

JavaScript では `new` 演算子を使用してオブジェクトを作成します。`Spry.Data.XMLDataSet` メソッドは、Spry データライブラリ内にあるコンストラクタで、新しい Spry データセットオブジェクトを作成します。このコンストラクタはパラメータを 2 つ受け取ります。1 つはデータのソース (この場合は相対 URL の "`data/cafetownsend.xml`"), もう 1 つは、データの提供元となる XML ノードを指定する XPath 式 ("`specials/menu_item`") です。

次のように、XML データのソースとして絶対 URL を指定することもできます。

```
var dsSpecials = new Spry.Data.XMLDataSet("http://www.somesite.com/somefolder/cafetownsend.xml",  
"specials/menu_item");
```

**注意：**絶対 URL か相対 URL かに関係なく、使用する URL はブラウザのセキュリティモデルの影響を受けます。したがって、リンク元の HTML ページと同じサーバードメインにある XML ソースからしかデータをロードすることはできません。この制限を回避するには、クロスドメインサービススクリプトを指定します。詳細については、サーバー管理者に確認してください。

上の例のコンストラクタでは、dsSpecials という新しい Spry データセットオブジェクトが作成されます。このデータセットは、"cafetownsend.xml" ファイルの specials/menu\_item ノード (XPath で指定されたノード) からデータを取得して、テーブルの行と列のような、オブジェクトとプロパティの平面化された配列に変換します (このテーブルの例については、このセクションの冒頭を参照してください)。

各データセットでは現在の行が維持されます。デフォルトでは、現在の行はデータセットの最初の行に設定されます。現在の行を後からプログラムで変更するには、データセットオブジェクトの setCurrentRow() メソッドを呼び出します。詳細については、109 ページの「現在の行の設定または変更」を参照してください。

**注意：**JavaScript 演算子 new で作成したデータセットにはデータはまだ含まれていません。データセットにデータをロードするには、まず、データセットの loadData() メソッドを呼び出します。このメソッドは、XML データをロードするための要求を実行します。Spry 領域と詳細領域では依存先のデータセットに対してこの操作が自動的に行われますが、これらの領域を使用しない場合は、ページコードで loadData() メソッドを手動で呼び出す必要があります。データは非同期的にロードされるため、loadData() を呼び出した直後にデータにアクセスしようとしてもアクセスできない場合があります。

## Spry XML データセットの高度な例

Spry XML データセットは、XMLHttpRequest オブジェクトを使用して、指定された URL を非同期的にロードします。XML データは、実際にはテキストフォーマットと DOM (ドキュメントオブジェクトモデル) ツリーフォーマットの 2 つのフォーマットでロードされます。

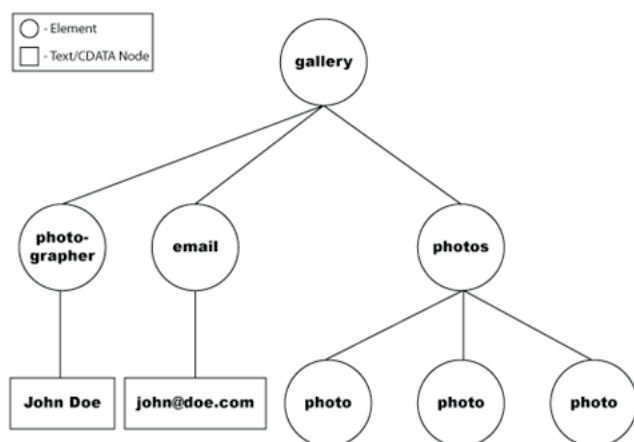
たとえば、データソースとして "\_/photos.php?galleryid=2000" を指定したとします (これは、XML データを取得する Web サービスへのパスです)。

```
<script type="text/javascript">  
    var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");  
</script>
```

次のコードは、テキストフォーマットでロードされたデータを表しています。

```
<gallery id="12345">  
    <photographer id="4532">John Doe</photographer>  
    <email>john@doe.com</email>  
    <photos id="2000">  
        <photo path="sun.jpg" width="16" height="16" />  
        <photo path="tree.jpg" width="16" height="16" />  
        <photo path="surf.jpg" width="16" height="16" />  
    </photos>  
</gallery>
```

次の例は、DOM ツリーフォーマットでロードされたデータを表しています。

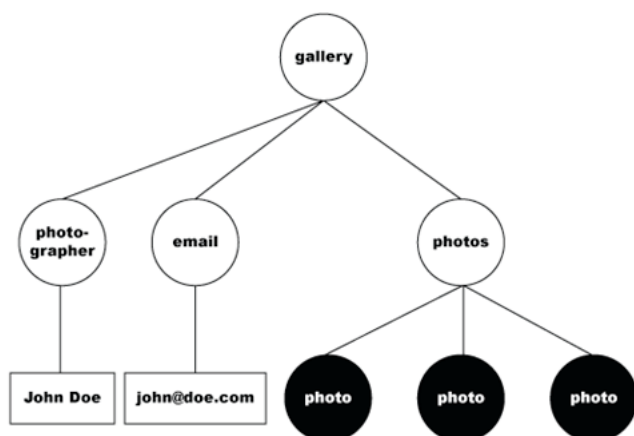


データがロードされると、データセットは、コンストラクタで指定された XPath を使用して XML DOM ツリーを移動し、目的のデータを表す特定のノードを見つけます。

次のコードでは、`/gallery/photos/photo` という XPath で選択されたデータがボードで示されています。

```
<gallery id="12345">
  <photographer id="4532">John Doe</photographer>
  <email>john@doe.com</email>
  <photos id="2000">
    <photo path="sun.jpg" width="16" height="16"/>
    <photo path="tree.jpg" width="16" height="16"/>
    <photo path="surf.jpg" width="16" height="16"/>
  </photos>
</gallery>
```

選択されたノードの DOM ツリー表現は次のようになります。



その後、ノードのセットが次のテーブルのようなフォーマットに平面化されます。

| @path    | @width | @height |
|----------|--------|---------|
| sun.jpg  | 16     | 16      |
| tree.jpg | 16     | 16      |
| surf.jpg | 16     | 16      |

この例では、平面化されたテーブルの列名が、選択されたノードとその属性から生成されています。ただし、列名がどのように決定されるかは、指定する XPath によって異なります。

データの平面化と列の作成では、次のガイドラインが使用されます。

- 選択したノードに属性がある場合は、各属性の列が作成されて、その列に属性の値が配置されます。それらの列の名前は、属性名の先頭に @ 記号を付けた名前になります。たとえば、ノードに id 属性があった場合、その列の名前は @id になります。
- 選択したノードにエレメントの子がなく、テキストや CDATA が含まれている場合は、列が作成されて、その列にそのテキストや CDATA が配置されます。この列の名前は、標準 XML エレメントのノードのタグ名になります。
- 選択したノードにエレメントの子がある場合は、各エレメントとその属性の値の列が作成されます。ただし、列が作成されるのは、自身はエレメントの子を持たないエレメントの子だけです。列の名前は子エレメントのタグ名になります。子エレメントの属性の場合は、"childTagName/@attrName" というフォーマットが使用されます。
- 選択したノードが属性の場合は、その属性の列が作成されます。列の名前は、属性名の先頭に @ 記号を付けた名前になります。
- 自身がエレメントの子を持つエレメントの子は無視されます。

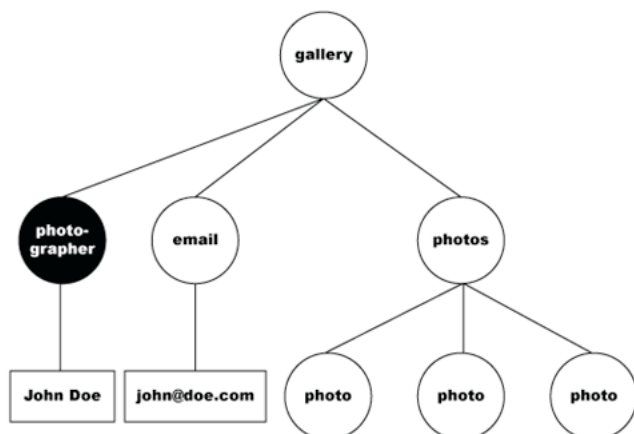
以降では、平面化のプロセスとデータセットの列名の生成方法をより詳しく示す例を紹介します。

#### 属性とテキスト値を含むエレメントを選択して平面化する例

次のコードでは、/gallery/photographer という XPath で選択されたデータがボードで示されています。

```
<gallery id="12345">
  <photographer id="4532">John Doe</photographer>
  <email>john@doe.com</email>
  <photos id="2000">
    <photo path="sun.jpg" width="16" height="16" />
    <photo path="tree.jpg" width="16" height="16" />
    <photo path="surf.jpg" width="16" height="16" />
  </photos>
</gallery>
```

選択されたノードの DOM ツリー表現は次のようになります。



選択されたデータは次のようなテーブルに平面化されます。

| photographer | @id |
|--------------|-----|
| John Doe     | 16  |

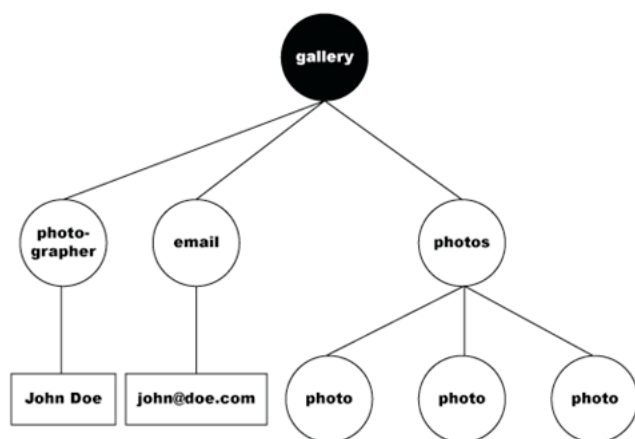
この例の場合、選択されているノードは 1 つだけなので、データセットに含まれる行は 1 行だけです。photographer ノードの値は "John Doe" というテキストなので、**photographer** という名前の列が作成されて、その値が格納されます。id 属性は photographer ノードの属性なので、この属性の値は @id 列に配置されます。すべての属性名の先頭に @ 記号が付きます。

#### 属性とエレメントの子を含むエレメントを選択して平面化する例

次のコードは、/gallery という XPath で選択されたデータを示しています。

```
<gallery id="12345">
  <photographer id="4532">John Doe</photographer>
  <email>john@doe.com</email>
  <photos id="2000">
    <photo path="sun.jpg" width="16" height="16" />
    <photo path="tree.jpg" width="16" height="16" />
    <photo path="surf.jpg" width="16" height="16" />
  </photos>
</gallery>
```

選択されたノードの DOM ツリー表現は次のようになります。



選択されたデータは次のようなテーブルに平面化されます。

| @id   | photographer | photographer/@id | email        |
|-------|--------------|------------------|--------------|
| 12345 | John Doe     | 4532             | john@doe.com |

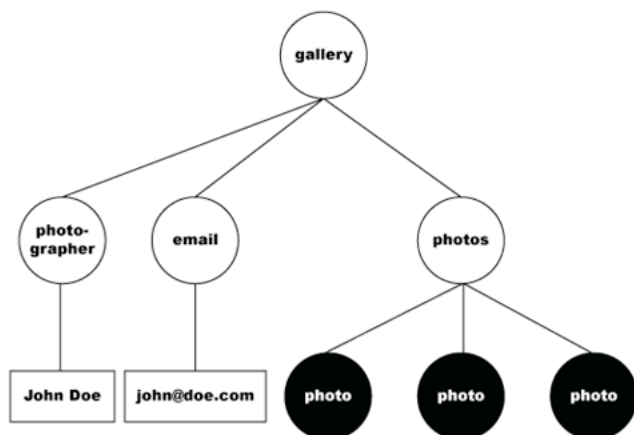
子エレメントの属性の列名には、先頭に子エレメントのタグ名が付いています。この例の場合、photographer は、選択された gallery ノードの子エレメントに当たるため、その id 属性の先頭には photographer/@ が付きます。また、photos エレメントは gallery ノードの子であるにもかかわらず、テーブルには追加されていません。これは、他のエレメントを含む子エレメントは平面化されないためです。

#### 1 つのエレメントの属性を選択して平面化する例

XPath ではノードの属性を選択することもできます。次のコードでは、gallery/photos/photo/@path という XPath で選択されたデータがボードで示されています。

```
<gallery id="12345">  
  <photographer id="4532">John Doe</photographer>  
  <email>john@doe.com</email>  
  <photos id="2000">  
    <photo path="sun.jpg" width="16" height="16" />  
    <photo path="tree.jpg" width="16" height="16" />  
    <photo path="surf.jpg" width="16" height="16" />  
  </photos>  
</gallery>
```

選択されたノードの DOM ツリー表現は次のようになります。



選択されたデータは次のようなテーブルに平面化されます。

|          |
|----------|
| @path    |
| sun.jpg  |
| tree.jpg |
| surf.jpg |

### Spry 動的領域の概要と構造

Spry データセットを作成したら、そのデータを Spry 動的領域に表示できます。Spry 動的領域とは、データセットにバインドされた Web ページ上の領域です。この領域にはデータセットの XML データが表示され、データセットが変更されるたびに自動的に表示が更新されます。

動的領域が再生成されるのは、バインド先のデータセットのオブザーバー（リスナー）として登録されているからです。データセットのデータが変更されると（データのロード、更新、ソート、フィルタ処理などが行われると）、そのデータセットのすべてのオブザーバーに通知が送信されて、リッスンしている動的領域が自動的に再生成されます。



コンテナタグで Spry 動的領域を宣言するには、`spry:region` 属性を使用します。ほとんどの HTML エレメントを動的領域のコンテナとして使用できますが、以下のタグは使用できません。

- col
- colgroup
- frameset
- html
- iframe
- select
- style
- table
- tbody
- tfoot
- thead
- title
- tr

これらの HTML エレメントは、Spry 動的領域のコンテナとしては使用できませんが、Spry 動的領域のコンテナの中で使用することはできます。

**注意:** 動的領域を使用できるのは `body` タグの中の領域だけです。`body` タグの外にあるタグに `spry:region` 属性を追加することはできません。

次の例では、`Specials_DIV` という動的領域のコンテナを、標準の HTML テーブルを含む `div` タグを使用して作成しています。テーブルは動的領域によく使用される HTML エレメントです。これは、テーブルでは 1 行目に見出しを、2 行目に XML データの繰り返しを格納できるからです。

```
<!--Create the Spry dynamic region-->
<div id="Specials_DIV" spry:region="dsSpecials">
  <!--Display the data in a table-->
  <table id="Specials_Table">
    <tr>
      <th>Item</th>
      <th>Description</th>
      <th>Price</th>
    </tr>
    <tr spry:repeat="dsSpecials">
      <td>{item}</td>
      <td>{description}</td>
      <td>{price}</td>
    </tr>
  </table>
</div>
```

この例では、動的領域のコンテナを作成する div タグに必要な属性は 2 つだけです。1 つは、動的領域を宣言し、そこで使用するデータセットを指定する spry:region 属性、もう 1 つは、領域の名前を指定する id 属性です。

```
<div id="Specials_DIV" spry:region="dsSpecials">
```

この新しい領域は、dsSpecials データセットのオブザーバーです。したがって、dsSpecials データセットが変更されるたびに、更新されたデータで再生成されます。

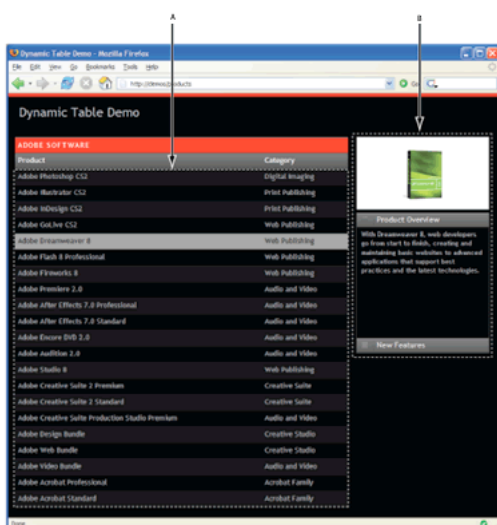
HTML テーブルには次のデータが表示されます。

```
<table id="Specials_Table">
  <tr>
    <th>Item</th>
    <th>Description</th>
    <th>Price</th>
  </tr>
  <tr spry:repeat="dsSpecials">
    <td>{item}</td>
    <td>{description}</td>
    <td>{price}</td>
  </tr>
</table>
```

テーブルの 2 行目の波カッコで囲まれている値 (データ参照) は、データセットの列を指定しています。これらのデータ参照により、テーブルセルがデータセットの特定の列のデータにバインドされます。XML データには繰り返しノードが含まれることが多いため、この例の 2 つ目のテーブル行タグでは spry:repeat 属性も宣言されています。これにより、ユーザーがページをロードすると、現在の行だけでなくデータセットのすべての行が表示されます。

## 基本的な Spry マスター / 詳細領域の概要と構造

Spry データセットを使用するには、マスター動的領域と詳細動的領域を作成すると、より詳細なデータを表示することができます。ページの一方向領域 (マスター領域) が、もう一方の領域 (詳細領域) に表示されるデータを制御します。



A. マスター領域 B. 詳細領域

通常、マスター領域にはデータセットの一連のレコードが簡略化された形で表示され、詳細領域には選択されたレコードに関するより詳細な情報が表示されます。詳細領域はマスター領域に依存するため、マスター領域のデータが変更されると詳細領域のデータも変更されます。

ここでは、両方の領域が同じデータセットに依存する基本的なマスター / 詳細関係について説明します。複数のデータセットを使用するマスター / 詳細領域については、91 ページの「高度な Spry マスター / 詳細領域の概要と構造」を参照してください。

次の例では、dsSpecials データセットのデータがマスター動的領域に表示され、マスター領域で選択されたデータの行に関する詳細情報が詳細動的領域に表示されます。

```
<head>
. . .
<script type="text/javascript" src="../includes/xpath.js"></script>
<script type="text/javascript" src="../includes/SpryData.js"></script>
<script type="text/javascript">
    var dsSpecials = new Spry.Data.XMLDataSet("data/cafetownsend.xml", "specials/menu_item");
</script>
</head>
. . .
<body>
<!--Create a master dynamic region-->
<div id="Specials_DIV" spry:region="dsSpecials">
    <table id="Specials_Table">
        <tr>
            <th>Item</th>
            <th>Description</th>
            <th>Price</th>
        </tr>
        <!--User clicks to reset the current row in the data set-->
        <tr spry:repeat="dsSpecials" spry:setrow="dsSpecials">
            <td>{item}</td>
```

```
        <td>{description}</td>
        <td>{price}</td>
    </tr>
</table>
</div>
<!--Create the detail dynamic region-->
<div id="Specials_Detail_DIV" spry:detailregion="dsSpecials">
    <table id="Specials_Detail_Table">
        <tr>
            <th>Ingredients</th>
            <th>Calories</th>
        </tr>
        <tr>
            <td>{ingredients}</td>
            <td>{calories}</td>
        </tr>
    </table>
</div>
. . .
</body>
```

**注意**：80 ページの「Spry XML データセットの基本的な概要」のサンプル XML ファイルには **Ingredients** ノードや **Calories** ノードは含まれていません。これらのノードは、この例のためにデータセットに追加されたものです。

この例では、1 つ目の div タグに id 属性と spry:region 属性が含まれています。これらの属性により、マスター動的領域のコンテナが作成されます。

```
<div id="Specials_DIV" spry:region="dsSpecials">
```

マスター領域の 1 つ目のテーブル行タグには、spry:setrow 属性が含まれています。この属性は、データセットの現在の行の値を設定します。

```
<tr spry:repeat="dsSpecials" spry:setrow="dsSpecials">
```

2 つ目の div タグには、詳細動的領域のコンテナを作成する属性が含まれています。

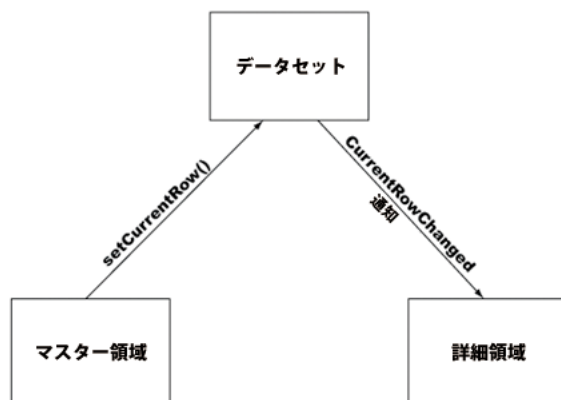
```
<div id="Specials_Detail_DIV" spry:detailregion="dsSpecials">
```

すべての Spry データセットで現在の行が維持されます。デフォルトでは、現在の行はデータセットの最初の行に設定されます。spry:detailregion は、データセットの現在の行が変更されると自動的に更新される以外は、spry:region とまったく同じように機能します。

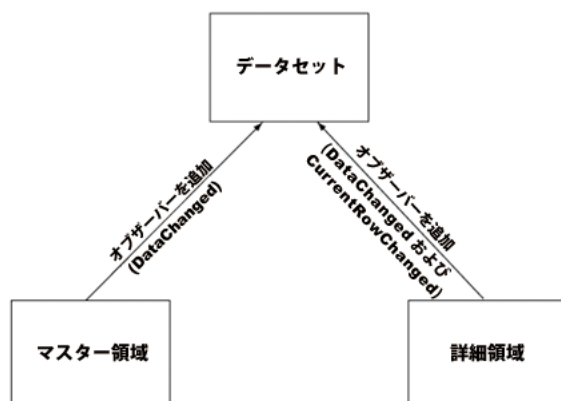
詳細領域では、バインディング式の {ingredients} および {calories} により、ページがブラウザにロードされるとデータセットの現在の行のデータが表示されます。しかし、ユーザーがマスター領域のテーブルの行をクリックすると、spry:setrow 属性により、データセットの現在の行がユーザーの選択した行に変更されます。

{ds\_RowID} データ参照は、Spry フレームワークの組み込み部分であり、データセットの各行に対して自動的に生成される一意の ID を参照します ( 詳細については、118 ページの「データ参照オプション」を参照してください)。ユーザーがマスター領域のテーブルの行を選択すると、spry:setrow 属性から setCurrentRow メソッドにその一意の ID が渡されて、データセットの現在の行が設定されます。

データセットが変更されるたびに、そのデータセットにバインドされているすべての動的領域が再生成されて、更新されたデータが表示されます。詳細領域もマスター領域と同様に dsSpecials データセットのオブザーバーになっているため、データセットが変更されると詳細領域も変更されて、ユーザーが選択した行（新しい現在の行）に関連するデータが表示されます。



`spry:region` と `spry:detailregion` の違いは次のとおりです。 `spry:detailregion` は、データセットからの `CurrentRowChange` の通知を (`DataChanged` の通知に加えて) リッスンしており、その通知を受信すると更新されます。一方、標準の `spry:regions` では、`CurrentRowChange` の通知は無視されます。この領域が更新されるのは、データセットから `DataChanged` の通知を受け取ったときだけです。



## 高度な Spry マスター / 詳細領域の概要と構造

複数のデータセットを含むマスター / 詳細関係を作成することもできます。たとえば、大量の詳細情報が関連付けられているメニュー品目のリストがあったとします（ここでは、説明のために材料のリストを使用します）。すべてのメニュー品目に関連付けられている情報をすべて1つのクエリーで取得すると、帯域幅がむだに使用されることになります。メニューのすべての品目の詳細情報に関心を持つユーザーはほとんどいないため、そうする必要もありません。必要な詳細情報のみをユーザーの求めに応じてダウンロードする方が効率的です。そうすれば、パフォーマンスも向上し、帯域幅も節約できます。このようにしてデータ交換の量を制限するのは、AJAX アプリケーションのパフォーマンスを向上させるために使用されている一般的なテクニックです。

以下は、"`cafetownsend.xml`" というサンプルファイルの XML ソースコードです。

```
<?xml version="1.0" encoding="UTF-8"?>
<specials>
  <menu_item id="1">
    <item>Summer Salad</item>
    <description>organic butter lettuce with apples, blood oranges, gorgonzola, and raspberry
vinaigrette.</description>
    <price>7</price>
    <url>summersalad.xml</url>
  </menu_item>
  <menu_item id="2">
    <item>Thai Noodle Salad</item>
    <description>lightly sauteed in sesame oil with baby bok choi, portobello mushrooms, and
scallions.</description>
    <price>8</price>
    <url>thainoodles.xml</url>
  </menu_item>
  <menu_item id="3">
    <item>Grilled Pacific Salmon</item>
    <description>served with new potatoes, diced beets, Italian parlsey, and lemon
zest.</description>
    <price>16</price>
    <url>salmon.xml</url>
  </menu_item>
</specials>
```

**注意：**このXML サンプルコードは、80 ページの「Spry XML データセットの基本的な概要」で使用したコードとは異なります。

"cafetownsend.xml" ファイルはマスターデータセットのデータを提供します。"cafetownsend.xml" ファイルの **url** ノードは、各メニュー品目の固有の XML ファイル ( または URL ) を参照しています。これらの固有の XML ファイルには、対応するメニュー品目の材料のリストが含まれています。たとえば、"summersalad.xml" ファイルの内容は次のようになっています。

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<item>
  <item_name>Summer salad</item_name>
  <ingredients>
    <ingredient>
      <name>butter lettuce</name>
    </ingredient>
    <ingredient>
      <name>Macintosh apples</name>
    </ingredient>
    <ingredient>
      <name>Blood oranges</name>
    </ingredient>
    <ingredient>
      <name>Gorgonzola cheese</name>
    </ingredient>
    <ingredient>
      <name>raspberries</name>
    </ingredient>
    <ingredient>
      <name>Extra virgin olive oil</name>
    </ingredient>
```

```
<ingredient>
  <name>balsamic vinegar</name>
</ingredient>
<ingredient>
  <name>sugar</name>
</ingredient>
<ingredient>
  <name>salt</name>
</ingredient>
<ingredient>
  <name>pepper</name>
</ingredient>
<ingredient>
  <name>parsley</name>
</ingredient>
<ingredient>
  <name>basil</name>
</ingredient>
</ingredients>
</item>
```

XML コードの構造がわかれば、マスター動的領域と詳細動的領域にデータを表示するために使用する 2 つのデータセットを作成できます。次の例では、dsSpecials データセットのデータがマスター動的領域に表示され、dsIngredients データセットのデータが詳細動的領域に表示されます。

```
<head>
. . .
<script type="text/javascript" src="../includes/xpath.js"></script>
<script type="text/javascript" src="../includes/SpryData.js"></script>
<script type="text/javascript">
<!--Create two separate data sets-->
  var dsSpecials = new Spry.Data.XMLDataSet("data/cafetownsend.xml", "specials/menu_item");
  var dsIngredients = new Spry.Data.XMLDataSet("data/{dsSpecials:url}",
"item/ingredients/ingredient");
</script>
</head>
. . .
<body>
<!--Create a master dynamic region-->
<div id="Specials_DIV" spry:region="dsSpecials">
  <table id="Specials_Table">
    <tr>
      <th>Item</th>
      <th>Description</th>
      <th>Price</th>
    </tr>
    <!--User clicks to reset the current row in the data set-->
```

```

    <tr spry:repeat="dsSpecials" spry:setrow="dsSpecials">
      <td>{item}</td>
      <td>{description}</td>
      <td>{price}</td>
    </tr>
  </table>
</div>
<!--Create the detail dynamic region-->
<div id="Specials_Detail_DIV" spry:region="dsIngredients">
  <table id="Specials_Detail_Table">
    <tr>
      <th>Ingredients</th>
    </tr>
    <tr spry:repeat="dsIngredients">
      <td>{name}</td>
    </tr>
  </table>
</div>
. . .
</body>

```

この例では、3つ目の script ブロックに、**dsSpecials** と **dsIngredients** という 2 つのデータセットを作成するステートメントが含まれています。

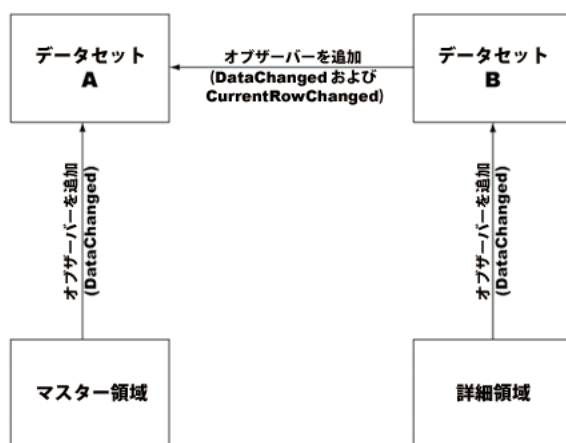
```

var dsSpecials = new Spry.Data.XMLDataSet("data/cafetownsend.xml", "specials/menu_item");
var dsIngredients = new Spry.Data.XMLDataSet("data/{dsSpecials:url}", "item/ingredients/ingredient");

```

2つ目のデータセットである **dsIngredients** の URL に含まれているデータ参照 (**{dsSpecials:url}**) は、1つ目のデータセットである **dsSpecials** へのデータ参照です。より具体的には、**dsSpecials** データセットの **url** 列へのデータ参照です。データセットを作成するコンストラクタの URL パラメータや XPath パラメータに別のデータセットへの参照が含まれている場合、作成されるデータセットは自動的に参照先データセットのオブザーバーになります。新しいデータセットは元のデータセットに依存しているため、元のデータセットのデータや現在の行が変更されるたびに、新しいデータセットのデータがリロードされたり、XPath が再適用されたりします。

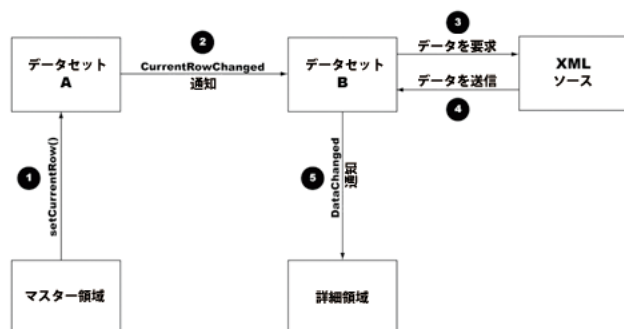
次の例は、データセットとマスターおよび詳細の各動的領域との間で確立されたオブザーバー関係を示しています。上の例では、**dsIngredients** データセット (データセット B) は **dsSpecials** データセット (データセット A) のオブザーバーです。



この例では、**dsSpecials** データセットの現在の行が変更されると、同じく変更が必要な **dsIngredients** データセットに通知が送信されます。**dsSpecials** データセットの各行の **url** 列にはそれぞれ異なる URL が含まれているため、選択された行の正しい URL が含まれるように **dsIngredients** データセットを更新する必要があります。

デフォルトでは、詳細領域に表示されるデータを含む **dsIngredients** データセットは、コンストラクタで指定された URL (この例の場合は **dsSpecials** データセットの **url** 列のデータへの参照) から取得したデータを使用して作成されます。**dsSpecials** データセットのデフォルトの現在の行 (最初の行) には "summersalad.xml" ファイルへの固有のパスが含ま

れているため、ページがブラウザにロードされたときには、そのファイルの情報が詳細領域に表示されます。しかし、dsSpecials データセットの現在の行が変更されると URL も (たとえば salmon.xml に) 変更されるため、それに応じて dsIngredients データセットが更新されます (その結果、このデータセットに関連付けられている詳細動的領域も更新されます)。



このプロセスは、88 ページの「基本的な Spry マスター / 詳細領域の概要と構造」で説明したプロセスと機能的には同等です。2つのプロセスの違いは、高度な例では2つ目の (詳細) データセットがマスターデータセットのデータと行の変更をリッスンしているのに対し、基本的な例では詳細領域がマスターデータセットのデータと行の変更をリッスンしているという点にあります。

コード例では、詳細領域に対して `spry:region` が使用されており、`spry:detailregion` は使用されていません。`spry:region` と `spry:detailregion` の違いは次のとおりです。`spry:detailregion` は、データセットからの `CurrentRowChange` の通知 (`DataChanged` の通知に加えて) リッスンしており、その通知を受信すると更新されます。`dsIngredients` データセットでは現在の行が変更されることはないため (変化するのは `dsSpecials` データセットの現在の行です)、`spry:detailregion` 属性は必要ありません。この場合は、`DataChanged` の通知のみをリッスンする領域を定義する `spry:region` 属性で十分です。

## プログレッシブエンハンスメントとデータセットのアクセシビリティについて

プログレッシブエンハンスメントでは、ブラウザの最小限の共通機能を対象にドキュメントのコードを作成し、CSS、JavaScript、Flash、Java、SVG コードなどを使用してページのプレゼンテーションや動作を強化します。この手法で作成すると、最新のブラウザでは高度な機能を利用できる一方で、それらのテクノロジーを利用できないブラウザもデータにアクセスでき、ページを利用できなくなることはありません。

このドキュメントでこれまでに紹介した Spry の例はすべて、JavaScript を使用して XML データを動的にロードして、ドキュメントの領域を生成する方法を扱ったものでした。Spry は、プログレッシブエンハンスメントの方式で使うこともできます。そのためには Hijax の手法を使用します。Hijax については、<http://domscripting.com/blog/display/41> を参照してください。このプログレッシブエンハンスメントの手法では、JavaScript を使用してページ上のアイテム (リンクなど) に目立たないようにイベントハンドラを関連付けて、ユーザーイベント (クリックなど) をキャッチします。プログレッシブエンハンスメントを使用すると、ドキュメントの一部をサーバーから非同期に配信されるコードに置き換えて、ページ全体をロードしなくても済むようにすることもできます。

では、この手法で Spry を使用する簡単な例を見てみましょう。次のような、静的データとリンクのみで構成された HTML ページから開始します。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:spry="http://ns.adobe.com/spry">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Hijax Demo - Notes 1</title>
</head>
<body>
<a href="notes1.html">Note 1</a>
<a href="notes2.html">Note 2</a>
<a href="notes3.html">Note 3</a>
<div>
  <p>This is some <b>static content</b> for note 1.</p>
</div>
</body>
</html>
```

このページで Spry をプログレッシブエンハンスメントの手法で使用して、リンクをクリックしたときにまったく新しいページをロードしなくても済むようにするには、まず、XML を使用して、リンクが参照している各ページのページフラグメントにアクセスできるようにします。次の例は、静的データを外部化する方法の一例を示しています。

```
<?xml version="1.0" encoding="iso-8859-1"?>
<notes>
  <note><![CDATA[<p>This is some <b>dynamic content</b> for note 1.</p>]]></note>
  <note><![CDATA[<p>This is some <b>dynamic content</b> for note 2.</p>]]></note>
  <note><![CDATA[<p>This is some <b>dynamic content</b> for note 3.</p>]]></note>
</notes>
```

その後、次のようにして Spry を HTML ページに適用できます。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:spry="http://ns.adobe.com/spry">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Hijax Demo - Notes 1</title>
<script language="JavaScript" type="text/javascript" src="includes/xpath.js"></script>
<script language="JavaScript" type="text/javascript" src="includes/SpryData.js"></script>
<script language="JavaScript" type="text/javascript">
<!--
var dsNotes = new Spry.Data.XMLDataSet('data/notes.xml', '/notes/note');
-->
</script>
</head>
<body>
<a href="note1.html" onclick="dsNotes.setCurrentRowNumber(0); return false;">Note 1</a>
<a href="note2.html" onclick="dsNotes.setCurrentRowNumber(1); return false;">Note 2</a>
<a href="note3.html" onclick="dsNotes.setCurrentRowNumber(2); return false;">Note 3</a>
<div spry:detailregion="dsNotes" spry:content="{note}">
  <p>This is some <b>static content</b> for note 1.</p>
</div>
</body>
</html>
```

上のコードに追加されている Spry コードは既に見慣れたものですが、div ブロックに spry:detailregion 属性だけでなく spry:content 属性も含まれています。この spry:content 属性は、Spry 動的領域の処理コードに対して、現在その領域に含まれている静的データを（その領域がバインドされているデータセットにデータがある場合に）属性値のデータ参照によって表されるデータに置き換えるように指定しています。

このページがロードされたブラウザで JavaScript が無効になっていた場合は、ページの機能が制限されて、静的コンテンツと従来のリンクナビゲーションからなる元のページと同じ機能が提供されます。JavaScript が有効になっていた場合は、データセットに XML データがロードされて、領域の静的コンテンツが置き換えられます。リンクをクリックすると、領域がデータセットのコードで更新されます。

上の例では、リンクが目立たないようにイベントハンドラを関連付けるために Hijax が利用されています。また、JavaScript のイベントハンドラを関連付ける趣旨をわかりやすくするために、意図的に onclick 属性を使用しています。

## Spry による動的ページの作成

### ファイルの準備

Spry データセットの作成を開始する前に必要なファイル ("xpath.js" および "SpryData.js") を入手してください。"xpath.js" ファイルは、データセットを作成するときに複雑な XPath 式を指定できるようにします。"SpryData.js" ファイルには Spry データライブラリが含まれています。

作成する HTML ページにこの両方のファイルをリンクします。

- 1 Adobe Labs の Web サイトで Spry ZIP ファイルを見つけます。
- 2 ハードドライブに Spry ZIP ファイルをダウンロードして解凍します。
- 3 解凍した "Spry" フォルダを開き、"includes" フォルダを見つけます。このフォルダに、Spry フレームワークを実行するために必要な "xpath.js" ファイルと "SpryData.js" ファイルが含まれています。
- 4 "includes" フォルダをコピーして Web サイトのルートディレクトリにペーストするか、"includes" フォルダのコピーを Web サイトのルートディレクトリにドラッグします。

**注意：**解凍した "Spry" フォルダから元の "includes" フォルダをドラッグして移動してしまうと、"Spry" フォルダ内のデモが正しく機能しなくなります。

- 5 コードビュー ([ 表示 ]-[ コード ]) で、ページのヘッドタグの中に以下の script タグを挿入して、Spry データライブラリのファイルを Web ページにリンクします。

```
<script type="text/javascript" src="includes/xpath.js"></script>
<script type="text/javascript" src="includes/SpryData.js"></script>
```

"SpryData.js" ファイルは "xpath.js" ファイルに依存しています。したがって、コード内で "xpath.js" ファイルの方が先にあることが重要です。

Spry データライブラリをリンクしたら、Spry データセットを作成できます。

- 6 Spry 名前空間宣言を HTML タグに追加します。HTML タグが次のようになります。

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:spry="http://ns.adobe.com/spry/">
```

Spry 名前空間宣言はコードを検査するために必要です。

**注意：**Spry データライブラリのファイルが HTML ページにリンクされている限り、Spry の機能はローカルで動作します。HTML ページをライブサーバーにパブリッシュするには、"xpath.js" ファイルと "SpryData.js" ファイルを依存ファイルとしてアップロードします。

### Spry XML データセットの作成

- 1 新しい HTML ページか既存の HTML ページを開きます。
- 2 Spry データライブラリのファイルがページにリンクされていること、および Spry 名前空間が宣言されていることを確認します。詳細については、97 ページの「ファイルの準備」を参照してください。
- 3 データセットの XML ソースを指定します。

"cafetownsend.xml" という XML ファイルを例として使用できます。このファイルは、サイトのルートフォルダ内の "data" というフォルダにあります。

data/cafetownsend.xml

次のように、XML ファイルへの URL を指定することもできます。

http://www.somesite.com/somefolder/cafetownsend.xml

**注意：**絶対 URL か相対 URL かに関係なく、使用する URL はブラウザのセキュリティモデルの影響を受けます。したがって、リンク元の HTML ページと同じサーバードメインにある XML ソースからしかデータをロードすることはできません。この制限を回避するには、クロスドメインサービススクリプトを指定します。詳細については、サーバー管理者に確認してください。

**4** データセットにデータを提供する繰り返し XML ノードを指定する必要があるため、データセットを作成する前に XML の構造を把握しておいてください。

次の例の "cafetownsend.xml" ファイルは、**specials** という親ノードと、そこに含まれる **menu\_item** という繰り返し子ノードで構成されています。

```
<?xml version="1.0" encoding="UTF-8"?>
<specials>
  <menu_item id="1">
    <item>Summer Salad</item>
    <description>organic butter lettuce with apples, blood oranges, gorgonzola, and raspberry vinaigrette.</description>
    <price>7</price>
  </menu_item>
  <menu_item id="2">
    <item>Thai Noodle Salad</item>
    <description>lightly sauteed in sesame oil with baby bok choy, portobello mushrooms, and scallions.</description>
    <price>8</price>
  </menu_item>
  <menu_item id="3">
    <item>Grilled Pacific Salmon</item>
    <description>served with new potatoes, diced beets, Italian parlsey, and lemon zest.</description>
    <price>16</price>
  </menu_item>
</specials>
```

**5** データセットを作成するには、次の script ブロックを、ライブラリを読み込む script タグの後に挿入します。

```
<script type="text/javascript">
  var datasetName = new Spry.Data.XMLDataSet("XMLsource", "XPathToRepeatingChildNode");
</script>
```

Cafe Townsend の例では、データセットを作成するステートメントは次のようになります。

```
var dsSpecials = new Spry.Data.XMLDataSet("data/cafetownsend.xml", "specials/menu_item");
```

このステートメントでは、dsSpecials という新しいデータセットが作成されます。このデータセットは、指定された XML ファイルの specials/menu\_item ノードからデータを取得します。このデータセットには、次のように、各メニュー品目に対応する行と、@id、item、description、および price の各列が含まれています。

| @id | item                   | description   | price |
|-----|------------------------|---|-------|
| 1   | Summer salad           | organic butter lettuce with apples, blood oranges, gorgonzola, and raspberry vinaigrette. | 7     |
| 2   | Thai Noodle Salad      | lightly sauteed in sesame oil with baby bok choy, portobello mushrooms, and scallions.    | 8     |
| 3   | Grilled Pacific Salmon | served with new potatoes, diced beets, Italian parlsey, and lemon zest.                   | 16    |

次のように、XML データのソースとして URL を指定することもできます。

```
var dsSpecials = new Spry.Data.XMLDataSet("http://www.somesite.com/somefolder/cafetownsend.xml",
"specials/menu_item");
```

**注意：**絶対 URL か相対 URL かに関係なく、使用する URL はブラウザのセキュリティモデルの影響を受けます。したがって、リンク元の HTML ページと同じサーバードメインにある XML ソースからしかデータをロードすることはできません。この制限を回避するには、クロスドメインサービススクリプトを指定します。詳細については、サーバー管理者に確認してください。

完成したコード例は次のようになります。

```
<head>
...
<script type="text/javascript" src="includes/xpath.js"></script>
<script type="text/javascript" src="includes/SpryData.js"></script>
<script type="text/javascript">
    var dsSpecials = new Spry.Data.XMLDataSet("data/cafetownsend.xml", "specials/menu_item");
</script>
...
</head>
```

**6 (オプション)** この例のようにデータセットの値に数字が含まれる場合は、それらの数値を含む列のタイプを再設定します。これは、後にデータをソートする場合に重要になります。

列のタイプを設定するには、次のボールドの部分に示すように、データセットの `setColumnType` メソッドをドキュメントのヘッドタグ ( のデータセットを作成した後 ) に追加します。

```
<script type="text/javascript">
    var dsSpecials = new Spry.Data.XMLDataSet("data/cafetownsend.xml", "specials/menu_item");
    dsSpecials.setColumnType("price", "number");
</script>
```

上の例の式は、既に定義されている `dsSpecials` データセットオブジェクトの `setColumnType` メソッドを呼び出します。`setColumnType` メソッドはパラメータを2つ受け取ります。1つは、タイプを変更するデータセットの名前 ("price")、もう1つは、目的のデータタイプ ("number") です。

詳細については、101 ページの「ユーザーによるデータのソート」を参照してください。

データセットを作成できたら、動的領域を作成してデータを表示できるようにします。

## Spry 動的領域の作成とデータの表示

Spry データセットを作成したら、Spry 動的領域をそのデータセットにバインドします。Spry 動的領域は、データが表示されるページ上の領域です。この領域では、データセットが変更されるたびに自動的に表示が更新されます。

**1** Spry データライブラリのファイルがページにリンクされていること、Spry 名前空間が宣言されていること、およびデータセットが作成されていることを確認します。詳細については、97 ページの「ファイルの準備」および 97 ページの「Spry XML データセットの作成」を参照してください。

**2** コードビューで、領域を挿入するタグに `spry:region` 属性を追加して、Spry 動的領域を作成します。この属性は、`spry:region="datasetName"` というシンタックスを使用します。

**注意：**すべてではありませんが、ほとんどの HTML エLEMENT は動的領域のコンテナとして使用できます。詳細については、86 ページの「Spry 動的領域の概要と構造」を参照してください。

たとえば、`div` タグを、`dsSpecials` データセットからのデータを表示する動的領域のコンテナとして使用するには、次のように `spry:region` 属性をタグに追加します。

```
<div id="Specials_DIV" spry:region="dsSpecials"></div>
```

**注意：**動的領域は複数のデータセットに依存することができます。データセットを領域に追加するには、追加するデータセットを `spry:region` 属性の追加の値として、スペースで区切って列記します。たとえば、`spry:region="dsSpecials dsSpecials2 dsSpecials3"` を使用して動的領域を作成できます。

**3** 動的領域のあるタグ内 ( この例では `div` タグを使用しています ) に、HTML エLEMENT を挿入してデータセットの最初の行を表示します。任意の HTML エLEMENT を使用してデータを表示できます。ただし、この目的で使用される最も標準的なELEMENT の1つは、2行の HTML テーブルです。この HTML テーブルでは、最初の行に静的な列見出しが格納され2行目にデータが格納されます。

```
<table id="Specials_Table">
  <tr>
    <th>Item</th>
    <th>Description</th>
    <th>Price</th>
  </tr>
  <tr spry:repeat="dsSpecials">
    <td>{item}</td>
    <td>{description}</td>
    <td>{price}</td>
  </tr>
</table>
```

2行目の波カッコで囲まれている値(データ参照)は、データセットの列を指定しています。これらのデータ参照により、テーブルセルがデータセットの特定の列のデータにバインドされます。

**注意:** Spry 領域が複数のデータセットに依存している場合は、動的領域のバインド先のデータセットを指定します。データ参照の完全なシンタックスは、{datasetName::columnName} という形式で表されます。たとえば、前の例のデータを使用して動的領域を複数の異なるデータセットにバインドするには、{dsSpecials::item}, {dsSpecials::description}, などのように入力します。Spry 領域も複数のデータセットをサポートします。複数のデータセットを指定するには、データセット名をスペースで区切って属性の値に追加します。形式は <div spry:region="hds1 ds2 ds3"> のようになります。

**4** HTML エlementを自動的に繰り返してデータセットのすべての行を表示するには、次のシンタックスを使用して、spry:repeat 属性と値を HTML エlementのタグに追加します。

```
spry:repeat="datasetName"
```

この例では、次のように spry:repeat 属性をテーブル行タグに追加します(ボールド部分)。

```
<tr spry:repeat="dsSpecials">
  <td>{item}</td>
  <td>{description}</td>
  <td>{price}</td>
</tr>
```

この例で、動的領域をデータセットにバインドするコードが完成すると次のようになります。

```
<div id="Specials_DIV" spry:region="dsSpecials">
  <table id="Specials_Table">
    <tr>
      <th>Item</th>
      <th>Description</th>
      <th>Price</th></tr>
    <tr spry:repeat="dsSpecials">
      <td>{item}</td>
      <td>{description}</td>
      <td>{price}</td>
    </tr>
  </table>
</div>
```

**5** ユーザーがデータをソートできるようにするためのクリックイベントを定義して、動的領域をさらにインタラクティブにすることができます。手順については、101 ページの「ユーザーによるデータのソート」を参照してください。

## サンプルコード: Spry データセットと動的領域

次のサンプルコードでは、Spry データセットと動的領域を作成して、HTML テーブル内に特殊なメニューのリストを表示します。

```
<head>
...
<script type="text/javascript" src="includes/xpath.js"></script>
<script type="text/javascript" src="includes/SpryData.js"></script>
<script type="text/javascript">
    var dsSpecials = new Spry.Data.XMLDataSet("data/cafetownsend.xml", "specials/menu_item");
</script>
...
</head>
<body>
...
<div id="Specials_DIV" spry:region="dsSpecials">
    <table id="Specials_Table">
        <tr>
            <th>Item</th>
            <th>Description</th>
            <th>Price</th>
        </tr>
        <tr spry:repeat="dsSpecials">
            <td>{item}</td>
            <td>{description}</td>
            <td>{price}</td>
        </tr>
    </table>
</div>
...
</body>
```

## ユーザーによるデータのソート

spry:sort 属性を動的領域に追加すると、ユーザーがデータを操作できるようになります。この項では、例として 99 ページの「Spry 動的領域の作成とデータの表示」のテーブルコードを使用します。

**1** spry:sort 属性 (複数可) を追加するコード内の場所を探します。この例では、属性を XML データを表示するテーブル内の 2 つの列ヘッダーに追加しています。

**2** 次のフォームを使用して、spry:sort 属性を適切な列ヘッダータグに追加します。

```
spry:sort="columnName"
```

spry:sort 属性で定義される値によって、データをソートするときに使用する列をデータセットに示します。

たとえば、次の spry:sort 属性 (ボールド部分) を列ヘッダータグに追加すると、ユーザーがページ上の列ヘッダーをクリックするたびに、指定された値に従って動的領域のデータがソートされます。

```
<div id="Specials_DIV" spry:region="dsSpecials">
    <table id="Specials_Table" class="main">
        <tr>
            <th spry:sort="item">Item</th>
            <th spry:sort="description">Description</th>
            <th>Price</th>
        </tr>
        <tr spry:repeat="dsSpecials">
            <td>{item}</td>
            <td>{description}</td>
            <td>{price}</td>
        </tr>
    </table>
</div>
```

ページの [Item] をクリックするとメニュー品目名に従ってデータがアルファベット順にソートされ、ページの [Description] をクリックすると、メニュー品目の詳細に従ってデータがアルファベット順にソートされます。

### 数値順のソート

デフォルトでは、データセット内のすべてのデータ（数値も含む）がテキストと見なされ、アルファベット順にソートされます。数値順にソートするには（メニュー品目の価格順にソートする場合など）、`setColumnType` データセットメソッドを使用して `price` 列のデータタイプをテキストから数値に変更できます。このメソッドは次の形式で表されます。

```
datasetName.setColumnType("columnName", "number");
```

前の例の場合、データセット（ボールド）を作成した後に、`setColumnType` メソッドをドキュメントのヘッドタグに追加します。

```
<script type="text/javascript">
  var dsSpecials = new Spry.Data.XMLDataSet("data/cafetownsend.xml", "specials/menu_item");
  dsSpecials.setColumnType("price", "number");
</script>
```

式は、`setColumnType` メソッドを、既に定義されている `dsSpecials` データセットオブジェクトで呼び出します。`setColumnType` メソッドはパラメータを 2 つ受け取ります。1 つは、タイプを変更するデータセットの名前（"price"）、もう 1 つは、目的のデータタイプ（"number"）です。

`spry:sort` 属性を `price` 列に追加して、ユーザーがいずれかのテーブルヘッダーをクリックしたときに HTML テーブル内の 3 つの列をすべてソートできるようにすることができます。

```
<div id="Specials_DIV" spry:region="dsSpecials">
  <table id="Specials_Table" class="main">
    <tr>
      <th spry:sort="item">Item</th>
      <th spry:sort="description">Description</th>
      <th spry:sort="price">Price</th>
    </tr>
    <tr spry:repeat="dsSpecials">
      <td>{item}</td>
      <td>{description}</td>
      <td>{price}</td>
    </tr>
  </table>
</div>
```

### 基本的なマスターページと詳細ページの作成

Spry データセットを使用するには、マスター動的領域と詳細動的領域を作成すると、より詳細なデータを表示することができます。ページの一方の領域（マスター領域）が、もう一方の領域（詳細領域）に表示されるデータを制御します。

基本的なマスター動的領域と詳細動的領域の動作の概要については、88 ページの「基本的な Spry マスター / 詳細領域の概要と構造」を参照してください。

- 1 データセットを作成します。97 ページの「Spry XML データセットの作成」を参照してください。
- 2 `spry:region` 属性を、領域のコンテナタグとして使用する HTML エレメントに追加してマスター領域を作成します。99 ページの「Spry 動的領域の作成とデータの表示」を参照してください。

次の例では、マスター動的領域に、`dsSpecials` データセットからの繰り返しのデータを表示します。

```

<head>
. . .
<script type="text/javascript" src="../../includes/xpath.js"></script>
<script type="text/javascript" src="../../includes/SpryData.js"></script>
<script type="text/javascript">
    var dsSpecials = new Spry.Data.XMLDataSet("data/cafetownsend.xml", "specials/menu_item");</script>
</head>
. . .
<body>
    <div id="Specials_DIV" spry:region="dsSpecials">
        <table id="Specials_Table">
            <tr>
                <th>Item</th>
                <th>Description</th>
                <th>Price</th>
            </tr>
            <tr spry:repeat="dsSpecials">
                <td>{item}</td>
                <td>{description}</td>
                <td>{price}</td>
            </tr>
        </table>
    </div>
</body>

```

**3** データセット内の現在の行をユーザーが変更できるようにするための属性を追加します。次の例では、ユーザーがマスター領域テーブル内の行をクリックするたびに、`spry:setrow` 属性 ( ボールド部分 ) によってデータセット内の現在の行が変更されます。

```

<tr spry:repeat="dsSpecials" spry:setrow="ds_Specials">
    <td>{item}</td>
    <td>{description}</td>
    <td>{price}</td>
</tr>

```

**4** `spry:detailregion` 属性を、領域を挿入するタグに追加してページに詳細動的領域を作成します。この属性は、`spry:detailregion="datasetName"` というシンタックスを使用します。

次の例では、`div` タグに、詳細動的領域があります。

```

<div id="Specials_Detail_DIV" spry:detailregion="dsSpecials">
</div>

```

**5** 詳細動的領域のあるタグ内に、HTML エlement を挿入してデータセットの現在の行の詳細データを表示します。

この例では、HTML テーブルに、`{ingredients}` 列および `{calories}` 列 ( `dsSpecials` データセット内 ) の詳細データが表示されます。

```

<div id="Specials_Detail_DIV" spry:detailregion="dsSpecials">
    <table id="Specials_Detail_Table">
        <tr>
            <th>Ingredients</th>
            <th>Calories</th>
        </tr>
        <tr>
            <td>{ingredients}</td>
            <td>{calories}</td>
        </tr>
    </table>
</div>

```

マスター動的領域と詳細動的領域の両方を `dsSpecials` データセットにバインドするコードが完成すると、次のようになります。

```

<div id="Specials_DIV" spry:region="dsSpecials">
  <table id="Specials_Table">
    <tr>
      <th>Item</th>
      <th>Description</th>
      <th>Price</th>
    </tr>
    <tr spry:repeat="dsSpecials" spry:setrow="dsSpecials">
      <td>{item}</td>
      <td>{description}</td>
      <td>{price}</td>
    </tr>
  </table>
</div>
<div id="Specials_Detail_DIV" spry:detailregion="dsSpecials">
  <table id="Specials_Detail_Table">
    <tr>
      <th>Ingredients</th>
      <th>Calories</th>
    </tr>
    <tr>
      <td>{ingredients}</td>
      <td>{calories}</td>
    </tr>
  </table>
</div>

```

## 高度なマスターページおよび詳細ページの作成

複数のデータセットに関連するマスター / 詳細関係を作成できます。このような関係の機能の概要については、91 ページの「高度な Spry マスター / 詳細領域の概要と構造」を参照してください。

- 1 データセットの作成で使用した XML ファイルの構造を学習します。あるデータセットを別のデータセットに依存させるためには構造を理解する必要があります。
- 2 ドキュメントの `head` に適切なコードを追加してデータセットを作成します (97 ページの「Spry XML データセットの作成」を参照してください)。これが、マスターデータセットになります。
- 3 作成したマスターデータセットのすぐ後に 2 番目のデータセット (詳細データセット) を作成します。詳細データセットのコンストラクタ内の URL または XPath には、マスターデータセット内の複数の列へのデータ参照があります。データ参照は、`{MasterDatasetName::columnName}` というシンタックスを使用します。

次の例では、3 番目の `script` ブロックに、**dsSpecials** (マスター) と **dsIngredients** (詳細) という名前の 2 つのデータセットを作成するステートメントが記述されています。

```

<head>
  . . .
  <script type="text/javascript" src="../../includes/xpath.js"></script>
  <script type="text/javascript" src="../../includes/SpryData.js"></script>
  <script type="text/javascript">
    var dsSpecials = new Spry.Data.XMLDataSet("data/cafetownsend.xml", "specials/menu_item");
    var dsIngredients = new Spry.Data.XMLDataSet("data/{dsSpecials::url}",
      "item/ingredients/ingredient");
  </script>
</head>

```

**dsIngredients** 詳細データセットの XML ファイルへのパスには、データ参照 (`{dsSpecials::url}`) があります。これは **dsSpecials** マスターデータセットに対する参照です。より具体的には、**dsSpecials** データセットの `url` 列へのデータ参照です。データセットを作成するコンストラクタ内の `url` または XPath パラメータに別のデータセットへの参照がある場合、作成されるデータセットは自動的に参照先データセットのオブザーバーになります。

- 4 `spry:region` 属性を、領域のコンテナタグとして使用する HTML エレメントに追加してマスター領域を作成します。99 ページの「Spry 動的領域の作成とデータの表示」を参照してください。

次の例では、マスター動的領域に、**dsSpecials** データセットからの繰り返しのデータを表示します。

```
<body>
  <div id="Specials_DIV" spry:region="dsSpecials">
    <table id="Specials_Table">
      <tr>
        <th>Item</th>
        <th>Description</th>
        <th>Price</th>
      </tr>
      <tr spry:repeat="dsSpecials">
        <td>{item}</td>
        <td>{description}</td>
        <td>{price}</td>
      </tr>
    </table>
  </div>
</body>
```

**5** マスターデータセット内の現在の行をユーザーが変更できるようにするための属性を追加します。次の例では、ユーザーがマスター領域テーブル内の行をクリックするたびに、`spry:setrow` 属性 ( ボールド部分 ) によって `dsSpecials` データセット内の現在の行が変更されます。

```
<tr spry:repeat="dsSpecials" spry:setrow="dsSpecials">
  <td>{item}</td>
  <td>{description}</td>
  <td>{price}</td>
</tr>
```

**6** `spry:region` 属性を、領域を挿入するタグに追加して、ページに詳細動的領域を作成します。この属性は、`spry:region="datasetName"` というシンタックスを使用します。

**注意**：2 つ以上のデータセットを使用してマスター / 詳細関係を作成する場合は、`spry:detailregion` 属性を使用する必要はありません。概要については、102 ページの「基本的なマスターページと詳細ページの作成」を参照してください。詳細データセットの現在の行は変更されないため ( 変更されるのはマスターデータセットの現在の行です )、`spry:region` 属性で十分です。詳細については、91 ページの「高度な Spry マスター / 詳細領域の概要と構造」を参照してください。

次の例では、`div` タグに、詳細動的領域があります。

```
<div id="Specials_Detail_DIV" spry:region="dsSpecials">
</div>
```

**7** 詳細動的領域のあるタグ内に、HTML エlement を挿入してマスターデータセットの現在の行の詳細データを表示します。

この例では、HTML テーブルに、`{name}` 列 (`dsIngredients` データセット内) の詳細データが表示されます。`dsIngredients` データセットは `{name}` 列を、`dsSpecials` データセットから受け取った情報を基に作成します。

```
<div id="Specials_Detail_DIV" spry:region="dsIngredients">
  <table id="Specials_Detail_Table">
    <tr>
      <th>Ingredients</th>
    </tr>
    <tr spry:repeat="dsIngredients">
      <td>{name}</td>
    </tr>
  </table>
</div>
```

マスター領域を `dsSpecials` データセットにバインドし、詳細領域を `dsIngredients` データセットにバインドするコード例が完成すると、次のようになります。

```
<head>
. . .
<script type="text/javascript" src="../../includes/xpath.js"></script>
<script type="text/javascript" src="../../includes/SpryData.js"></script>
<script type="text/javascript">
    var dsSpecials = new Spry.Data.XMLDataSet("data/cafetownsend.xml", "specials/menu_item");
    var dsIngredients = new Spry.Data.XMLDataSet("data/{dsSpecials::url}",
"item/ingredients/ingredient");
</script>
</head>
. . .
<body>
    <div id="Specials_DIV" spry:region="dsSpecials">
        <table id="Specials_Table">
            <tr>
                <th>Item</th>
                <th>Description</th>
                <th>Price</th>
            </tr>
            <tr spry:repeat="dsSpecials" spry:setrow="dsSpecials">
                <td>{item}</td>
                <td>{description}</td>
                <td>{price}</td>
            </tr>
        </table>
    </div>
    <div id="Specials_Detail_DIV" spry:region="dsIngredients">
        <table id="Specials_Detail_Table">
            <tr>
                <th>Ingredients</th>
            </tr>
            <tr spry:repeat="dsIngredients">
                <td>{name}</td>
            </tr>
        </table>
    </div>
. . .
</body>
```

## データの取得と操作

### データセット取得オプション

デフォルトでは、Spry データセットは HTTP GET メソッドを使用して XML データを取得します。次のように追加のコンストラクタオプションを指定すると、HTTP POST メソッドを使用してデータを取得することもできるようになります。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Dynamic Region Example</title>
<script type="text/javascript" src="../../includes/xpath.js"></script>
<script type="text/javascript" src="../../includes/SpryData.js"></script>
<script type="text/javascript">
  var dsPhotos = new Spry.Data.XMLDataSet("/photos.php", "/gallery/photos/photo", { method: "POST",
postData: "galleryid=2000&offset=20&limit=10", headers: { "Content-Type": "application/x-www-form-
urlencoded; charset=UTF-8" } });
</script>
</head>
. . .
<body>
</body>
</html>
```

POST メソッドの使用時にコンテンツタイプを指定しなかった場合、デフォルトのコンテンツタイプは "application/x-www-form-urlencoded; charset=UTF-8" に設定されます。

次の表は、指定可能な HTTP 関連のコンストラクタオプションを示しています。

| オプション    | 説明  |
|----------|---|
| method   | XML データを取得するときに使用する HTTP メソッド。文字列 "GET" または "POST" を指定する必要があります。  |
| postData | url エンコードのフォームパラメータを含む文字列、または XMLHttpRequest オブジェクトでサポートされる任意の値を指定できます。"Content-Type" ヘッダーが postData オプションと組み合わせて指定されていない場合、"application/x-www-form-urlencoded; charset=UTF-8" コンテンツタイプが使用されます。 |
| username | XML データにアクセスするときに使用するサーバーのユーザー名。  |
| password | XML データにアクセスするときに username と組み合わせて使用するパスワード。  |
| headers  | 値を保存するプロパティおよびキーとして HTTP 要求フィールド名を使用するオブジェクトまたは連想配列。  |

### データキャッシュのオフ

デフォルトでは、クライアントでデータセットにロードしたすべての XML データがキャッシュされます。データセットがキャッシュ内に既に存在する特定の URL の XML データをロードしようとする時、Spry はデータセットにキャッシュされたデータへの参照を返します。複数のデータセットが同時に同じ URL をロードしようとする時、すべてのロード要求が 1 つの HTTP 要求に結合されて帯域幅が節約されます。

データキャッシュと要求の結合機能を使用すると、特に複数のデータセットが同じ XML データを参照している場合やサーバーからデータを直接ロードする必要がある場合（ユーザーがアクセスするたびに別のデータを返す URL がある場合など）に、パフォーマンスが大幅に向上します。

❖ データセットでサーバーから直接 XML データをロードするには、次のように useCache XMLDataSet コンストラクタオプションを false に設定します。

```
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo",
{useCache: false })
```

### データの取得

XML データは、ロードされた後、テーブルのようなフォーマットに平面化されます。データセット内のデータは、実際にはプロパティ (列) と値を持つオブジェクト (行) の配列として保存されます。

次の例では、/gallery/photos/photo という XPath で選択されたデータがボードで示されています。

```
<gallery id="12345">
  <photographer id="4532">John Doe</photographer>
  <email>john@doe.com</email>
  <photos id="2000">
    <photo path="sun.jpg" width="16" height="16"/>
    <photo path="tree.jpg" width="16" height="16"/>
    <photo path="surf.jpg" width="16" height="16"/>
  </photos>
</gallery>
```

その後、ノードのセットが次のテーブルのようなフォーマットに平面化されます。

| @path    | @width | @height |
|----------|--------|---------|
| sun.jpg  | 16     | 16      |
| tree.jpg | 16     | 16      |
| surf.jpg | 16     | 16      |

`getData()` を呼び出すとデータセット内のすべての行を取得できます。データは非同期にロードされるため、ロード後のデータにのみアクセスできます。データの準備ができたときに通知を受け取るオブザーバーを登録する必要がある場合があります。詳細については、110 ページの「オブザーバー通知の使用」を参照してください。

❖ `getData()` メソッドを使用して、データセット内のすべての行を取得します。特定の列の値を取得するには、列名を使用して行のインデックスを作成します。

```
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");
...
var rows = dsPhotos.getData(); // Get all rows.
var path = rows[0]["@path"]; // Get the data in the "@path" column of the first row.
```

## データのソート

❖ 特定の列の値を使用してデータセットの行をソートするには、データセットで `sort()` メソッドを呼び出します。

```
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");
...
dsPhotos.sort("@path"); // Sort the rows of the data set using the "@path" column.
```

データセットの `sort()` メソッドを呼び出すと、指定された列のデータを使用して行が昇順にソートされます。

`sort()` メソッドはパラメータを2つ受け取ります。最初のパラメータには、ソート時に使用する列の名前または列名の配列を指定できます。最初のパラメータが配列の場合、配列の最初のエレメントはプライマリソート列として機能し、それ以降の各列は、2 番目、3 番目、およびそれ以降のソートに使用されます。2 番目のパラメータは、使用するソート順序です。"ascending"、"descending"、または "toggle" のいずれかの文字列を指定する必要があります。2 番目のパラメータを省略した場合、ソート順序はデフォルトの "ascending" になります。ソート順序として "toggle" を指定すると、"ascending" 順でソートされたデータが "descending" 順にソートされます。場合によってはその逆になることもあります。列が初めてソートされる場合にソート順序として "toggle" が指定されると、データはまず "ascending" 順でソートされます。

```
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");
...
dsPhotos.sort("@path", "toggle"); // Toggle the Sort order of the rows of the data set using the "@path" column.
```

データをロードするたびにデータセットのデータが自動的にソートされるようにするには、ソートの基準となる列と、データセットコンストラクタに対するオプションとして使用するソート順序を指定します。データセットコンストラクタに対して "sortOnLoad" オプションを使用して、ソートの基準にする列を指定します。次の例では、データセットは自動的に、@path 列のデータを使用して降順にソートされます。

```
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo", {
  sortOnLoad: "@path", sortOrderOnLoad: "descending" });
```

デフォルトでは、データセットのすべての値がテキストとして扱われます。これは、数値または日付値をソートするときの問題になる場合があります。setColumnType() メソッドを呼び出して、特定の列のデータタイプを指定できます。

```
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");
dsPhotos.setColumnType("@width", "number");
dsPhotos.setColumnType("@height", "number");
...
dsPhotos.sort("@width"); // Sort the rows of the data set using the "@width" column.
```

現在サポートされているデータタイプは、"number"、"date"、および"string"です。

### 現在の行の設定または変更

各データセットでは現在の行が維持されます。デフォルトでは、現在の行はデータセットの最初の行に設定されます。現在の行をプログラムで変更するには、`setCurrentRowNumber()` メソッドを呼び出して、現在の行として設定する行の行番号を渡します。最初の行のインデックスは常に 0 であるため、データセットの行数が 10 の場合、最後の行のインデックスは 9 になります。

❖ `setCurrentRowByNumber()` または `setCurrentRow()` を使用して、現在の行を変更します。

```
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");
...
dsPhotos.setCurrentRowNumber(2); // Make the 3rd row in the data set the current row.
```

データセットの各行には一意の ID が割り当てられます。この ID を使用すると、データセットの行の順序が変更された後でも、データセットの特定の行を参照できます。行の ID は、その行の "ds\_RowID" プロパティにアクセスすると取得できます。`setCurrentRow()` を呼び出して、現在の行として設定する行の行 ID を渡すことによって、現在の行を変更することもできます。

```
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");
...
var id = dsPhotos.getData()[2]["ds_RowID"]; // Get the ID of the 3rd row.
...
dsPhotos.setCurrentRow(id); // Make the 3rd row the current row by using its ID.
```

### 重複行の削除

❖ `distinct()` メソッドを使用してデータセットの重複した行を削除します。

```
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");
...
dsPhotos.distinct(); // Remove all duplicate rows.
```

この場合、重複行は、データセットの各列に同じ情報の行が 2 行以上存在する状況を意味します。

データがデータセットにロードされるたびに `distinct()` メソッドが自動的に実行されるようにするには、コンストラクタに対して "distinctOnLoad" オプションを指定します。

```
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo", {
distinctOnLoad: true });
```

`distinct()` メソッドは破壊的であるため、同じ行はすべて破棄されます。破棄されたデータを元に戻す唯一の方法は、XML データをリロードすることです。

### データのフィルタ処理

データセットでは、破壊的なフィルタと非破壊的なフィルタの両方をサポートします。

いずれのフィルタ方法も、使用する前にデータセット、行オブジェクトおよび `rowNumber` を取得するフィルタ関数を指定します。この関数は、データセットの各行に対するデータセットフィルタメソッドで呼び出されます。また、この関数に渡される行オブジェクト、または渡された行を置き換えるための新しい行オブジェクトを返す必要があります。フィルタにより行を除外する関数の場合は、`null` 値を返す必要があります。

データセットの破壊的なフィルタメソッドは、`filterData()` です。このメソッドは実際にはデータセットの行を置き換えるか、破棄します。元のデータを復元する唯一の方法は、データセットの XML データをリロードすることです。

❖ 破壊的な `filterData()` メソッドを使用して、データセットの行を完全に破棄します。

```

...
// Filter out all rows that don't have a path that begins
// with the letter 's'.
var myFilterFunc = function(dataSet, row, rowNumber)
{
    if (row["@path"].search(/^s/) != -1)
        return row; // Return the row to keep it in the data set.
    return null; // Return null to remove the row from the data set.
}dsPhotos.filterData(myFilterFunc); // Filter the rows in the data set.

```

フィルタ関数は、別の URL から XML データをロードしても、`filterData()` を `null` パラメータを指定して呼び出すまではアクティブなままになります。`filterData()` を、`null` パラメータを使用して呼び出すと、フィルタ関数をアンインストールできます。

```
dsPhotos.filterData(null); // Turn off destructive filtering.
```

データセットの非破壊的なフィルタメソッドは、`filter()` です。`filterData()` とは異なり、`filter()` は、元のデータを参照する新しい行の配列を作成します。フィルタ関数が渡された行オブジェクトを変更しない限り、`filter()` を `null` パラメータを渡して呼び出すことで元のデータを取得できます。非破壊的な `filter()` メソッドを使用して、データセットの行をフィルタ処理します。

```

var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");
...
// Filter out all rows that don't have a path that begins
// with the letter 's'.
var myFilterFunc = function(dataSet, row, rowNumber)
{
    if (row["@path"].search(/^s/) != -1)
        return row; // Return the row to keep it in the data set.
    return null; // Return null to remove the row from the data set.
}dsPhotos.filter(myFilterFunc); // Filter the rows in the data set.

```

元のデータを復元するには、`filter()` を呼び出し、`null` パラメータを渡します。

```
dsPhotos.filter(null); // Turn off non-destructive filtering.
```

## データの更新

データセットは、ミリ秒単位で指定した間隔でデータをリロードできます。これは、特定の URL にあるデータが定期的に変更される場合に便利です。

❖ データセットに特定の区間でロードするよう指定するには、オプションの `loadInterval` を渡して、`XMLDataSet` コンストラクタを呼び出します。

```
// Load the data every 10 seconds. Turn off the cache to make sure we get it directly from the server.
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo", {
    useCache: false, loadInterval: 10000 });
```

また、プログラムでこの区間でのロードを有効にするには `startLoadInterval()` メソッドを、ロードを停止するには `stopLoadInterval()` メソッドを使用することもできます。

```

dsPhotos.startLoadInterval(10000); // Start loading data every 10 seconds.
...
dsPhotos.stopLoadInterval(); // Turn off interval loading.

```

## オブザーバー通知の使用

### オブザーバー通知の概要

XML データセットは、オブジェクトまたはコールバック関数がイベント通知を受信するためのオブザーバーメカニズムをサポートします。

| 通知                  | 説明  |
|---------------------|---|
| onPreLoad           | データセットが、データの要求を送信しようとしています。データセットが他のデータセットに依存している場合、このイベント通知は、すべてが正常にロードされるまで送信されません。 |
| onPostLoad          | データの要求は成功しました。データにアクセスできます。   |
| onLoadError         | データの要求中にエラーが発生しました。   |
| onDataChanged       | データセットのデータが修正されています。  |
| onPreSort           | データセットのデータをソートしようとしています。  |
| onPostSort          | データセットのデータはソートされています。   |
| onCurrentRowChanged | 現在の行のデータセットの概念が変更されています。  |

### オブザーバーとしてのオブジェクト

通知を受信するには、オブジェクトは受信に関心がある通知ごとに1つのメソッドを定義し、オブジェクト自体をデータセット上のオブザーバーとして登録します。たとえば、onDataChanged 通知に関心があるオブジェクトは、onDataChanged() メソッドを定義して addObserver() を呼び出します。

```
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");
...
var myObserver = new Object;
myObserver.onDataChanged = function(dataSet, data)
{
    alert("onDataChanged called!");
};
dsPhotos.addObserver(myObserver);
```

各通知メソッドの最初のパラメータは、通知を送信しているオブジェクトになります。データセットオブザーバーの場合、これは常に dataSet オブジェクトになります。2番目のパラメータは、未定義か、通知の種類に依存しているオブジェクトになります。

| 通知                  | 通知に渡されるデータ                                      |
|---------------------|---|
| onPreLoad           | 未定義   |
| onPostLoad          | 未定義   |
| onLoadError         | 要求を行うときに使用された Spry.Utils.loadURL.Request オブジェクト |
| onDataChanged       | 未定義   |
| onPreSort           | 次のプロパティを使用するオブジェクト                              |
|                     | oldSortColumns: 前回のソートに使用された列の配列                |
|                     | oldSortOrder: 前回のソートに使用されたソート順序                 |
|                     | newSortColumns: 次回のソートに使用される列の配列                |
|                     | newSortOrder: 次回のソートに使用されるソート順序                 |
| onPostSort          | 次のプロパティを使用するオブジェクト                              |
|                     | oldSortColumns: 前のソートに使用された列の配列                 |
|                     | oldSortOrder: 前のソートに使用されたソート順序                  |
|                     | newSortColumns: ソートに使用される列の配列                   |
|                     | newSortOrder: ソートに使用されるソート順序                    |
| onCurrentRowChanged | 次のプロパティを使用するオブジェクト                              |
|                     | oldRowID: 直前の現在の行の ds_RowID                     |
|                     | newRowID: 現在の行の ds_RowID                        |

オブジェクトで通知の受信を停止するには、`removeObserver()` を呼び出してオブザーバーのリストからオブジェクトを削除する必要があります。

```
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");  
...  
dsPhotos.removeObserver(myObserver);
```

### オブザーバーとしての関数

関数をオブザーバーとして登録することもできます。オブジェクトと関数オブザーバーの主な違いは、オブジェクトは、それが定義している通知メソッドについての通知のみを受けますが、関数オブザーバーはすべての種類のイベント通知に対して呼び出されるという点です。

```
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");  
...  
function myObserverFunc(notificationType, dataSet, data)  
{  
    if (notificationType == "onDataChanged")  
        alert("onDataChanged called!");  
    else if (notificationType == "onPostSort")  
        alert("onPostSort called!");  
};  
dsPhotos.addObserver(myObserverFunc);
```

関数オブザーバーは、`addObserver` と同じ呼び出しで登録されます。

関数が呼び出される時、その関数に渡される最初のパラメータは通知の種類になります。これは、通知の名前を示すストリングです。2 番目のパラメータは通知側で、この場合はデータセットになります。3 番目のパラメータは通知のデータです。

関数オブザーバーで通知の受信を停止するには、`removeObserver()` を呼び出してオブザーバーのリストからその関数オブザーバーを削除する必要があります。

```
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");  
...  
dsPhotos.removeObserver(myObserverFunc);
```

## 動的領域の使用

### ループ構造

動的領域は現在 2 つのループ構造をサポートしています。一方の構造では、特定のデータセット (`spry:repeat`) の行ごとに 1 つの要素とそのコンテンツをすべて繰り返し、もう一方では、特定のデータセット (`spry:repeatchildren`) の行ごとに特定の要素の子をすべて繰り返すことができます。

要素を繰り返すよう指定するには、`spry:repeat` 属性を、繰り返すデータセット名の要素に追加します。次の例は、`li` ブロックを示しています。このブロックは、`dsPhotos` データセットのすべての行に対して繰り返されます。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:spry="http://ns.adobe.com/spry">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Dynamic Region Example</title>
<script type="text/javascript" src="../../includes/xpath.js"></script>
<script type="text/javascript" src="../../includes/SpryData.js"></script>
<script type="text/javascript">
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");
</script>
</head>
<body>
  <div spry:region="dsPhotos">
    <ul>
      <li spry:repeat="dsPhotos">{@path}</li>
    </ul>
  </div>
</body>
</html>
```

エレメントの子だけを繰り返すには、`spry:repeatchildren` 属性を使用します。次の例では、`ul` タグの子が、`dsPhotos` データセットのすべての行に対して繰り返されます。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:spry="http://ns.adobe.com/spry">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Dynamic Region Example</title>
<script type="text/javascript" src="../../includes/xpath.js"></script>
<script type="text/javascript" src="../../includes/SpryData.js"></script>
<script type="text/javascript">
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");
</script>
</head>
<body>
  <div spry:region="dsPhotos">
    <ul spry:repeatchildren="dsPhotos">
      <li>{@path}</li>
    </ul>
  </div>
</body>
</html>
```

前の `"spry:repeat"` と `"spry:repeatchildren"` の例は機能的には同じですが、`"spry:repeatchildren"` は条件構造と組み合わせて使用するとさらに便利です。いずれの例でも、結果は次のよう出力されます。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:spry="http://ns.adobe.com/spry">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Dynamic Region Example</title>
<script type="text/javascript" src="../../includes/xpath.js"></script>
<script type="text/javascript" src="../../includes/SpryData.js"></script>
<script type="text/javascript">
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");
</script>
</head>
<body>
<div>
<ul>
<li>sun.jpg</li>
<li>tree.jpg</li>
<li>surf.jpg</li>
</ul>
</div>
</body>
</html>
```

データセットのすべての行に対する繰り返し領域内のコンテンツを出力しない場合は、`spry:test` 属性を、`spry:repeat` またはその上にある `spry:repeatchildren` 属性を持つエレメントに追加してループ処理中に出力される内容を制限します。

次の例は、`li` ブロックを示しています。このブロックは、`{@path}` の値の最初の文字が `s` で始まる場合にのみ出力されます。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:spry="http://ns.adobe.com/spry">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Dynamic Region Example</title>
<script type="text/javascript" src="../../includes/xpath.js"></script>
<script type="text/javascript" src="../../includes/SpryData.js"></script>
<script type="text/javascript">
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");
</script>
</head>
<body>
<div spry:region="dsPhotos">
<ul>
<li spry:repeat="dsPhotos" spry:test="{@path}'.search(/^s/) != -1;">{@path}</li>
</ul>
</div>
</body>
</html>
```

この `spry:test` 属性の値は、ゼロまたは `false` あるいはゼロ以外の値に評価される JavaScript 式になります。式がゼロ以外の値を返す場合、コンテンツが出力されます。XHTML を使用している場合、`&`、`<`、`>` など JavaScript 式で使用される可能性のある特殊文字を HTML エンティティに変換する必要があります。この JavaScript 式内のデータ参照を使用して、動的領域処理エンジンで、`spry:test` 式を評価する直前にデータセットから実際の値を提供することもできます。

次のコードは、前の例の最終的な出力を示しています。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:spry="http://ns.adobe.com/spry">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Dynamic Region Example</title>
<script type="text/javascript" src="../../includes/xpath.js"></script>
<script type="text/javascript" src="../../includes/SpryData.js"></script>
<script type="text/javascript">
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");
</script>
</head>
<body>
  <div>
    <ul>
      <li>sun.jpg</li>
      <li>surf.jpg</li>
    </ul>
  </div>
</body>
</html>
```

## 条件構造

動的領域は現在 2 つの条件構造をサポートしています。1 つは "spry:if" です。次の例の li タグは、{@path} の値の最初の文字が **s** で始まる場合にのみ出力されます。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:spry="http://ns.adobe.com/spry">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Dynamic Region Example</title>
<script type="text/javascript" src="../../includes/xpath.js"></script>
<script type="text/javascript" src="../../includes/SpryData.js"></script>
<script type="text/javascript">
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");
</script>
</head>
<body>
  <div spry:region="dsPhotos">
    <ul class="spry:repeat">
      <li spry:if="'{@path}'.search(/^[s/]) != -1;">{@path}</li>
    </ul>
  </div>
</body>
</html>
```

エレメントに条件を付けるには、spry:if 属性を、ゼロまたはゼロ以外の値を返す JavaScript 式の値を持つエレメントに追加します。JavaScript 式がゼロ以外の値を返すと、エレメントが最終的な出力に書き込まれます。

if-else 構造が必要な場合は、"spry:choose" 構造を使用します。次の例では、spry:choose 構造を使用して、他のすべての div を特徴付けます。

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:spry="http://ns.adobe.com/spry">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Dynamic Region Example</title>
<script type="text/javascript" src="../../includes/xpath.js"></script>
<script type="text/javascript" src="../../includes/SpryData.js"></script>
<script type="text/javascript">
var dsPhotos = new Spry.Data.XMLDataSet("/photos.php?galleryid=2000", "/gallery/photos/photo");
</script>
</head>
<body>
<div spry:region="dsPhotos">
<div spry:choose="spry:choose">
<div spry:when="{@path}" == 'surf.gif'>{@path}</div>
<div spry:when="{@path}" == 'undefined'>Path was not defined.</div>
<div spry:default="spry:default">Unexpected value for path!</div>
</div>
</div>
</body>
</html>

```

spry:choose 構造には、case ステートメントや if-else if-else 構造と同じ機能があります。spry:choose 構造を作成するには、spry:choose 属性をエレメントに追加します。次に、spry:when 属性を持つ子エレメントを追加します。子エレメントは複数追加できます。spry:when 属性の値は、ゼロまたはゼロ以外の値を返す JavaScript 式にする必要があります。デフォルトで、各 spry:when 属性の JavaScript 式がすべてゼロまたは false を返すようにする場合は、spry:default 属性を持つエレメントを追加します。spry:default 属性には値は必要ありませんが、XHTML ではすべての属性に値が必要となるため、属性の値をその名前と同じ値に設定します。

領域処理エンジンは、各ノードの spry:when 属性を、その親エレメントの下にリストされている順に評価します。spry:default エレメントが常に最後に評価され、spry:when が無い場合に限り、式は 0 以外の値を返します。

## 領域の状態

Spry は、領域の状態の概念をサポートします。つまり、領域は、特定の時間にデータをロードしてデータを表示できる状態であるか、1 つまたは複数のデータセットがそのデータのロードに失敗しているためにエラー状態になります。

spry:state 属性を、"loading"、"error"、または "ready" の値を使用して領域コンテナ内のエレメントに配置して、特定の領域の状態に関連付けます。この操作を実行すると、領域のデータをロードする際にロード中のメッセージを表示する場合や、領域がそのデータの取得に失敗したことをユーザーに通知する場合に役立ちます。領域の状態が変化すると、そのコードが自動的に再生成されて、現在の状態と一致する spry:state 属性を持つエレメントが表示されます。

次の例では、spry:state 属性を使用してロード中のメッセージとエラーメッセージを表示します。

```

<div spry:region="dsEmployees">
<div spry:state="loading">Loading employee data ...</div>
<div spry:state="error">Failed to load employee data!</div>
<ul spry:state="ready">
<li spry:repeat="dsEmployees">{firstname} {lastname}</li>
</ul>
</div>

```

spry:state 属性を持っていないコンテンツ、または spry:state 属性を持っているエレメントの子または子孫ではないコンテンツは常に、コードの再生成時に出力に含まれます。また、spry:state 属性を持つエレメントの子または子孫は、spry:state 属性を持つことができません。つまり、spry:state 属性を持つエレメントをネストすることはできません。

## 領域オブザーバー通知

Spry は、開発者がオブジェクトまたは関数を登録して、領域の状態が変化するたびに通知を受け取れるようにするためのオブザーバーメカニズムをサポートしています。このメカニズムは、次の例外を除き、データセットで使用されるメカニズムとほとんど同じです。

- 領域オブザーバーの追加と削除は、`Spry.Data.Region.addObserver()` と `Spry.Data.Region.removeObserver` のグローバルな `namespaced` 関数で実行します。データセットオブザーバーはデータセットオブジェクト上の `addObserver()` メソッドと `removeObserver()` メソッドを呼び出すため、この方法はデータセットと異なります。グローバルな `namespaced` 関数を使用すると、ドキュメントの `onload` イベントが開始する前、および領域を表す JavaScript オブジェクトを Spry が作成する前に、開発者はオブザーバーを登録できます。開発者は JavaScript 領域が実際に存在する前にオブザーバーを登録する必要があるため、領域は、`addObserver` 関数と `removeObserver` 関数を使用します。
- `addObserver()` と `removeObserver()` には、開発者が監視する領域を識別するための ID が必要です。このため、開発者が監視する領域には、その領域コンテナノードで定義された `id` 属性が必要です。

## 領域オブザーバーとしてのオブジェクト

通知を受信するには、オブジェクトは受信に関心がある通知ごとに 1 つのメソッドを定義し、オブジェクト自体を領域上のオブザーバーとして登録します。

次の例は、動的領域でオブザーバーとして登録されているオブジェクトを示しています。

```
<script>
...
// Create an observer object and define the methods to receive the notifications
// it wants.
myObserver = new Object;
myObserver.onPostUpdate = function(notifier, data)
{
    alert("onPostUpdate called for " + data.regionID);
};
...
// Call addObserver() to register the object as an observer.
Spry.Data.Region.addObserver("employeeListRegion", myObserver);
...
// You can unregister your object so it stops receiving notifications
// with a call to removeObserver().
Spry.Data.Region.removeObserver("employeeListRegion", myObserver);
...
</script>
...
<ul id="employeeListRegion" spry:region="dsEmployees">
...
</ul>
```

各通知メソッドの最初のパラメータは、通知を送信しているオブジェクトになります。領域オブザーバーの場合、これは領域オブジェクトではありません。2 番目のパラメータは、通知をトリガした領域を識別する `regionID` プロパティを含むオブジェクトです。

オブジェクトで通知の受信を停止するには、`removeObserver()` を呼び出してオブザーバーのリストからオブジェクトを削除する必要があります。

## 領域オブザーバーとしての関数

関数をオブザーバーとして登録することもできます。オブジェクトと関数オブザーバーの主な違いは、オブジェクトは、それが定義している通知メソッドについての通知のみを受け取りますが、関数オブザーバーはすべての種類のイベント通知に対して呼び出されるという点です。

次の例は、動的領域でオブザーバーとして登録されている関数を示しています。

```

<script>
...
function myRegionCallback(notificationState, notifier, data)
{
  if (notificationType == "onPreUpdate")
    alert(regionID + " is starting an update!");
  else if (notificationType == "onPostUpdate")
    alert(regionID + " is done updating!");
}
...
// Call addObserver() to register your function as an observer.
Spry.Data.Region.addObserver("employeeListRegion", MyRegionCallback);
...
// You can unregister your callback function so it stops receiving notifications
// with a call to removeObserver().
Spry.Data.Region.removeObserver("employeeListRegion", MyRegionCallback);
...
</script>
...
<ul id="employeeListRegion" spry:region="dsEmployees">
...
</ul>

```

関数オブザーバーは、`addObserver()` と同じ呼び出しで登録されます。

関数が呼び出される時、その関数に渡される最初のパラメータは通知の種類になります。これは、通知の名前を示すストリングです。2 番目のパラメータは通知側で、この場合、領域オブジェクトではありません。3 番目のパラメータは、通知をトリガした領域を伝える `regionID` プロパティを持つデータオブジェクトです。

関数オブザーバーで通知の受信を停止するには、`removeObserver()` を呼び出してオブザーバーのリストからその関数オブザーバーを削除する必要があります。

次の表に、サポートされる通知の現在のセットを示します。

| 領域通知の種類                    | 説明   |
|----------------------------|--|
| <code>onLoadingData</code> | 領域のバインドデータセットの1つまたは複数とそのデータをロードしています。          |
| <code>onPreUpdate</code>   | 領域のバインドデータセットはすべて正常にロードされました。領域は、そのコードを再生成します。 |
| <code>onPostUpdate</code>  | 領域はそのコードを再生成し、ドキュメントに挿入しています。                  |
| <code>onError</code>       | データのロード中にエラーが発生しました。                           |

## データ参照オプション

各データセットには、動的領域の再生成プロセス中に使用できる組み込みのデータ参照のセットが含まれています。データセットの列名と同様に、動的領域が複数のデータセットにバインドされている場合は、これらの組み込みのデータ参照の前にデータセットの名前を付ける必要があります。

たとえば、領域の再生成時にデータセットの現在の行の行数を表示するには、次のように `ds_RowNumber` データ参照を動的領域に追加します。

```

<tr spry:repeat="dsSpecials">
  <td>{item}</td>
  <td>{description}</td>
  <td>{price}</td>
  <td>{ds_RowNumber}</td>
</tr>

```

これらのオプションは、次のように JavaScript メソッドで値を渡す場合にも役立ちます。

```
<tr spry:repeat="dsSpecials" onclick="dsSpecials.setCurrentRow('{ds_RowID}')">
  <td>{item}</td>
  <td>{description}</td>
  <td>{price}</td>
</tr>
```

次の表に、組み込みの Spry データ参照の完全なリストを示します。

| データ参照                 | 説明  |
|-----------------------|---|
| ds_RowID              | データセットに含まれる行の ID。この ID を使用して、データセット内の特定のレコードを参照できます。データがソートされた場合でも、ID は変更されません。                       |
| ds_RowNumber          | データセットに含まれる現在の行の行数。ループ構造内では、この数値は、現在評価されている行の位置を表します。   |
| ds_RowNumberPlus1     | ds_RowNumber と同じですが、最初の行はインデックス 0 ではなくインデックス 1 から始まります。   |
| ds_RowCount           | データセットに含まれている行の数。非破壊的なフィルタがデータセットに設定されている場合は、フィルタが適用された後の行の総数になります。                                   |
| ds_UnfilteredRowCount | 非破壊的なフィルタが適用される前にデータセットに含まれていた行数。   |
| ds_CurrentRowID       | データセットの現在の行の ID。この値は、ループ構造内で使用される場合でも変更されません。   |
| ds_CurrentRowNumber   | データセットに含まれる現在の行の行数。この値は、ループ構造内で使用される場合でも変更されません。  |
| ds_SortColumn         | ソートに最後に使用された列の名前。データセット内のデータがソートされない場合は、何も出力されません (空の文字列)。  |
| ds_SortOrder          | データセットに含まれるデータの現在のソート順序。このデータ参照は、 <b>ascending</b> または <b>descending</b> という語を出力するか、何も出力しません (空の文字列)。 |
| ds_EvenOddRow         | ds_RowNumber の現在の値を参照し、文字列 "even" または "odd" を返します。行の代替カラーを表示するのに役立ちます。                                |

## データ参照の非表示

ブラウザによっては、低速の接続でページをロードすると、ドキュメントの `onload` 通知が送信される前に、ページ上の未処理の領域やデータ参照をユーザーが簡単に参照できる場合があります。未処理の領域とデータ参照を非表示にするには、`SpryHiddenRegion` クラスの CSS ルールを指定します。

```
<style>.SpryHiddenRegion {visibility:hidden;}
</style>
...
<div spry:region="dsEmployees" class="SpryHiddenRegion">
...
</div>
```

この方法を使用すると、CSS ルールは、ページがロードされるときにこのクラスでマークされた Spry 領域を非表示にします。Spry データの処理が完了すると、Spry は `SpryHiddenRegion` クラスを取り除き、最終的な Spry コンテンツが表示されます。

タグ全体ではなく、データ参照だけを非表示にするには、`spry:content` 属性をデータ参照の代替として使用する方法があります。データ参照は `spry:content` 属性の値であるため、ページがロードされたときに表示されません。

次の例では、エレメントで `spry:content` 属性を使用してデータ参照を非表示にします。

```
<!--Example of a normal region.-->
<div spry:region="dsEmployees">
Hello my name is {firstname}.
</div>
<!--Example of a region using spry:content.-->
Hello my name is <span spry:content="{firstname}"></span>.
</div>
```

spry:content 属性は、div タグのコンテンツ全体を、属性の値で置き換えます。この例では、{firstname} の値が、空の span タグに挿入されます。結果は同じで、この例の場合にのみ表示可能なデータ参照がありません。

## ビヘイビア属性

動的領域内のエレメントにビヘイビア属性を配置して、通常は手動のプログラミングが必要になる共通のビヘイビアを自動的に有効にできます。

### spry:hover

spry:hover 属性は、マウスカーソルがエレメントに入るたびにエレメントにクラス名を配置し、カーソルがエレメントから離れるとそのクラス名を削除します。

spry:hover 属性の値は、マウスがエレメントに入るたび、またはエレメントから離れるたびにエレメントに置かれるクラスの名前です。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:spry="http://ns.adobe.com/spry">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Behavior Attributes Example</title>
<style>
.myHoverClass { background-color: yellow; }
</style>
<script type="text/javascript" src="../../includes/xpath.js"></script>
<script type="text/javascript" src="../../includes/SpryData.js"></script><script type="text/javascript">
var dsEmployees = new Spry.Data.XMLDataSet("../data/employees-01.xml",
"/employees/employee");
</script>
</head>
<body>
<div spry:region="dsEmployees">
  <ul>
    <li spry:repeat="dsEmployees" spry:hover="myHoverClass">{username}</li>
  </ul>
</div>
</body>
</html>
```

上の例では、マウスが li エレメントに入るたびに、“myHoverClass” クラス名がエレメントのクラス属性に追加されます。これは、マウスがエレメントから離れるときに自動的に削除されます。

### spry:select

spry:select 属性は、マウスでエレメントがクリックされると、エレメントにクラス名を配置します。

spry:select 属性の値は、マウスでエレメントがクリックされるたびにエレメントに置かれるクラスの名前です。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:spry="http://ns.adobe.com/spry">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Behavior Attributes Example</title>
<style>
.myHoverClass {
    background-color: yellow;
}
.mySelectClass {color: white;background-color: black;}
</style>
<script type="text/javascript" src="../../includes/xpath.js"></script>
<script type="text/javascript" src="../../includes/SpryData.js"></script><script
type="text/javascript">var dsEmployees = new Spry.Data.XMLDataSet("../../data/employees-01.xml",
"/employees/employee");
</script>
</head>
<body>
<div spry:region="dsEmployees">
    <ul>
        <li spry:repeat="dsEmployees" spry:hover="myHoverClass"
spry:select="mySelectClass">{username}</li>
    </ul>
</div>
</body>
</html>
```

上の例では、マウスがli要素をクリックするたびに、mySelectClass クラス名が要素のクラス属性に追加されます。

spry:select 属性を持つページ上の要素が以前に選択されている場合、その spry:select 属性の値として使用されたクラス名が自動的に削除され、その結果その要素は選択されません。

spry:selectgroup 属性を spry:select 属性と組み合わせると、ページ上に複数の選択のセットを持つことができます。この作業例については、Adobe Labs の "Spry" フォルダの "demos" フォルダにある RSS リーダーの例を参照してください。

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:spry="http://ns.adobe.com/spry">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Behavior Attributes Example</title>
<style>
.myHoverClass {
    background-color: yellow;
}
.mySelectClass {color: white;background-color: black;}
.myOtherSelectClass {color: white;background-color: black;}
</style>
<script type="text/javascript" src="../../includes/xpath.js"></script><script type="text/javascript"
src="../../includes/SpryData.js"></script><script type="text/javascript">
var dsEmployees = new Spry.Data.XMLDataSet("../../data/employees-01.xml",
"/employees/employee");
</script></head><body><div spry:region="dsEmployees">
    <ul>
        <li spry:repeat="dsEmployees" spry:hover="myHoverClass" spry:select="mySelectClass"
spry:selectgroup="username">{username}</li></ul>
        <li spry:repeat="dsEmployees" spry:hover="myHoverClass" spry:select="myOtherSelectClass"
spry:selectgroup="firstname">{firstname}</li>
    </ul>
</div>
</body></html>
```

spry:selectgroup 属性の値は任意の名前です。その spry:selectgroup 属性と同じ名前を使用する要素は、同じ選択グループ名の別の要素がクリックされると自動的に選択解除されます。spry:selectgroup 値が異なる他の要素は影響を受けません。

## 第 4 章：Spry 効果の使用

### Spry 効果について

#### Spry 効果について

効果は、HTML ページのほぼすべてのエレメントに適用できる視覚効果です。たとえば、情報のハイライト、移行アニメーションの作成、特定の期間のページエレメントの視覚的な変更などに使用されます。効果は、Web サイトの外観と印象を向上する簡単で優れた方法です。

効果ではクライアントにある JavaScript のメソッドおよび関数が参照されるため、使用する場合にサーバーサイドのロジックまたはスクリプトは必要ありません。したがって、ユーザーが HTML ページを参照して効果を起動した場合は、効果が適用されるオブジェクトのみが更新されます。ページ全体を更新する必要はありません。

Spry framework for AJAX には、次のような効果があります。

**フェード** エレメントを表示またはフェードアウトします。

**ハイライト** エレメントの背景色を変更します。

**ブラインドアップ/ブラインドダウン** 上下に動く窓用ブラインドのようにエレメントの表示 / 非表示を切り替えます。

**スライドアップ/スライドダウン** エレメントを上下に動かします。

**拡張** エレメントのサイズを拡大または縮小します。

**シェイク** 左から右へのエレメントの揺れをシミュレートします。

**スキッシュ** エレメントをページの左上隅に消します。

Spry 効果はページ上に簡単に実装できるように設計されており、実際の処理はフレームワークで行われます。新しいタグや馴染みのないシンタックスは必要としません。また、効果を適用する HTML エレメントで、カスタムタグを使用する必要はありません。

**注意：** 使用できる効果ライブラリはいくつかありますが、Adobe では、よくまとまったライブラリとしてコミュニティからの支持が厚い [Script.aculo.us](http://Script.aculo.us) を評価しています。したがって、これらの効果のリストおよび用語を採用するとともに、ブラウザの効果に関する問題について [Script.aculo.us](http://Script.aculo.us) のソリューションのいくつかを実装しています。また、Adobe では Thomas Fuchs 氏の [Script.aculo.us](http://Script.aculo.us) における実績も評価しており、どちらのライブラリも有益かつ有効なライブラリとしてコミュニティで認知されていると考えています。さらに、1 つのページで両方のライブラリを使用可能にするためのディベロッパーに向けた取り組みを行っています。

#### Spry 効果ライブラリについて

SpryEffects.js ファイル内にある Spry 効果ライブラリには、Adobe Labs で使用可能なすべての Spry 効果が含まれています。このファイルには、他の依存性はありません。

ページに効果を追加する場合は、HTML ドキュメントの先頭で次のように指定して "SpryEffects.js" ファイルをリンクします。

```
<script type="text/javascript" src="../../includes/SpryEffects.js"></script>
```

**注意：** 正確なファイルパスは、"SpryEffects.js" ファイルの格納場所によって異なります。

Spry 効果を使用するには、JavaScript ファイルと効果を含む HTML ファイルの両方をサーバー上に配置する必要があります。

## 概要

### ファイルの準備

Spry 効果を Web ページの要素に適用する前に、該当するファイルをダウンロードしてリンクします。

- 1 Adobe Labs の Web サイトで Spry ZIP ファイルを見つけます。
- 2 ハードドライブに Spry ZIP ファイルをダウンロードして解凍します。
- 3 解凍した "Spry" フォルダを開き、"includes" フォルダを見つけます。このフォルダには、Spry 効果を適用するために必要なファイルが格納されています。
- 4 次のいずれかの操作を行って、"SpryEffects.js" ファイルを Web サイトに追加します。
  - "includes" フォルダをコピーして、そのコピーを Web サイトのルートディレクトリにペーストまたはドラッグします。これにより、Spry 効果および Spry XML データセットの作成に必要なすべてのファイルを取得できます。
  - Web サイトでフォルダ (たとえば、"**SpryAssets**" という名前のフォルダ) を作成する場合は、"includes" フォルダを開き、新しいフォルダに "SpryEffects.js" ファイルをコピーします。

**注意:** 解凍した "Spry" フォルダから元の "includes" フォルダまたは個別ファイルをドラッグすると、"Spry" フォルダに含まれるデモが適切に動作しません。Web サイトにはドラッグするのではなく、コピー & ペーストしてください。

- 5 "SpryEffects.js" ファイルを Web サイトに含めると、これをリンクして Spry 効果をページに追加できるようになります。特定の効果をページに追加する具体的な手順については、それぞれの効果に関する項を参照してください。

## 効果の適用

### ハイライト効果の適用

ハイライト効果は、ターゲット要素の背景色を変更します。

ハイライト効果は、applet、body、frame、frameset、および noframes を除く任意の HTML 要素に適用できます。

- 1 "SpryEffects.js" ファイルを Web ページにリンクするには、次のコードをドキュメントの先頭に追加します。

```
<head>
. . .
<script src="../../includes/SpryEffects.js" type="text/javascript" ></script>
</head>
```

**注意:** 正確なファイルパスは、"SpryEffects.js" ファイルの格納場所によって異なります。

"SpryEffects.js" ファイルは、Adobe Labs からダウンロードした "Spry" フォルダの "includes" フォルダにあります。123 ページの「ファイルの準備」を参照してください。

- 2 ターゲット要素に、一意の ID が指定されていることを確認します。ユーザーがページで効果を生じさせるインタラクティブな操作を行うと、このターゲット要素が変化します。

```
<div class="demoDiv" id="highlight1"> Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum.</div>
```

- 3 効果を作成するには、ユーザーがページでインタラクティブな操作を行ったときに効果を生じさせる JavaScript イベントを追加します。たとえば、ユーザーが文をクリックしたときに別の段落をハイライトするには、次のイベントをその文の p タグに追加します。

```
<p><a onclick="Spry.Effect.DoHighlight('highlight1',{duration: 1000, from:'#CCCCCC', to:'#FFCC33',restoreColor: '#FFCC33',toggle:true}); return false;" href="#"> Click here to highlight the below paragraph.</a></p>
```

JavaScript メソッドの最初のパラメータは、常にターゲット要素の ID (上の例では 'highlight1') です。

コード全体を次に示します。

```

<head>
. . .
<script src="../../includes/SpryEffects.js" type="text/javascript" ></script>
</head>
<body>
<p><a onclick="Spry.Effect.DoHighlight('highlight1',{duration: 1000, from:'#CCCCCC',
to:'#FFCC33',restoreColor: '#FFCC33',toggle:true}); return false;" href="#"> Click here to highlight the
below paragraph.</a></p>
<div class="demoDiv" id="highlight1"> Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam
nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et
accusam et justo duo dolores et ea rebum.</div>
</body>

```

### ハイライト効果のオプション

次の表は、ハイライト効果で使用できるオプションのリストです。

| オプション      | 説明  |
|------------|---|
| duration   | 効果の継続時間をミリ秒単位で指定します。デフォルト値は 1000 です。  |
| from       | 開始カラー値を RGB カラー形式 (#RRGGBB) で指定します。この値は、ハイライトの先頭フレームの色を設定します。デフォルトは、ターゲットエレメントの背景色です。   |
| to         | 終了カラー値を RGB カラー形式 (#RRGGBB) で指定します。この値は、ハイライトの最終フレームの色を設定します。   |
| toggle     | 切り替え効果を生じます。デフォルト値は false です。この値を true に設定した場合は、restoreColor オプションは無視されます。  |
| transition | 移行のタイプとして linear または sinusoidal を指定します。linear では、移行が継続する間、移行速度が一定になります。sinusoidal では、効果がゆるやかに開始され、加速した後、再び減速して終了します。デフォルトは sinusoidal です。 |
| setup      | 効果の開始前に呼び出す関数を定義できます。たとえば、setup:function (element, effect) { /* ... */ } のように指定します。   |
| finish     | 効果の終了後に呼び出す関数を定義できます。たとえば、finish:function (element, effect) { /* ... */ } のように指定します。  |

サンプルコード：

```
Spry.Effect.DoHighlight('targetID', {duration:1000,from:'#00ff00',to:'#0000ff'});
```

### フェード効果の適用

フェード効果は、エレメントを表示またはフェードアウトします。

フェード効果は、applet、body、iframe、object、tr、tbody、および th を除く任意の HTML エレメントに適用できます。

**1** "SpryEffects.js" ファイルを Web ページにリンクするには、次のコードをドキュメントの先頭に追加します。

```

<head>
. . .
<script src="../../includes/SpryEffects.js" type="text/javascript" ></script>
</head>

```

**注意：** 正確なファイルパスは、"SpryEffects.js" ファイルの格納場所によって異なります。

"SpryEffects.js" ファイルは、Adobe Labs からダウンロードした "Spry" フォルダの "includes" フォルダにあります。123 ページの「ファイルの準備」を参照してください。

**2** ターゲットエレメントに、一意の ID が指定されていることを確認します。ユーザーがページで効果が発生させるインタラクティブな操作を行うと、このターゲットエレメントが変化します。

```

<div class="demoDiv" id="fade1">Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy
eirmod tempor invidunt ut labore et dolore magna sea takimata sanctus est Lorem ipsum dolor sit amet.</div>

```

**3** 効果を作成するには、ユーザーがページでインタラクティブな操作を行ったときに効果が発生させる JavaScript イベントを追加します。たとえば、ユーザーが文をクリックしたときに別の段落をフェードするには、次のイベントをその文の p タグに追加します。

```
<p><a onclick="Spry.Effect.DoFade('fade1', {duration:1000,from:100,to:20,toggle:true}); return false;" href="#"> Click here to make the paragraph fade from 100% to 20%.</a></p>
```

JavaScript メソッドの最初のパラメータは、常にターゲットエレメントの ID (上の例では 'fade1') です。

コード全体を次に示します。

```
<head>
. . .
<script src="../../includes/SpryEffects.js" type="text/javascript" ></script>
</head>
<body>
<p><a onclick="Spry.Effect.DoFade('fade1', {duration:1000,from:100,to:20,toggle:true}); return false;" href="#"> Click here to make the paragraph fade from 100% to 20%.</a></p>
<div class="demoDiv" id="fade1">Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna sea takimata sanctus est Lorem ipsum dolor sit amet.</div>
</body>
```

### フェード効果のオプション

次の表は、フェード効果で使用できるオプションのリストです。

| オプション      | 説明  |
|------------|---|
| duration   | 効果の継続時間をミリ秒単位で指定します。デフォルト値は 1000 です。  |
| from       | 透明度の開始値を % で指定します。デフォルト値は 0 です。   |
| to         | 透明度の終了値を % で指定します。デフォルト値は 100 です。   |
| toggle     | 切り替え効果を生成します。デフォルト値は false です。  |
| transition | 移行のタイプとして linear または sinusoidal を指定します。linear では、移行が継続する間、移行速度が一定になります。sinusoidal では、効果がゆるやかに開始され、加速した後、再び減速して終了します。デフォルトは sinusoidal です。 |
| setup      | 効果の開始前に呼び出す関数を定義できます。たとえば、setup:function (element, effect) { /* ... */ } のように指定します。   |
| finish     | 効果の終了後に呼び出す関数を定義できます。たとえば、finish:function (element, effect) { /* ... */ } のように指定します。  |

サンプルコード:

```
Spry.Effect.DoFade('targetID',{duration: 1000,from: 0,to: 100,toggle: true});
```

### ブラインドアップ / ブラインドダウン効果の適用

ブラインドアップ / ブラインドダウン効果は、上下に動く窓用ブラインドのようにエレメントの表示 / 非表示を切り替えます。この効果はスライド効果に似ていますが、コンテンツの位置は変わりません。

この効果を適用できる HTML エレメントは、address、dd、div、dl、dt、form、h1、h2、h3、h4、h5、h6、p、ol、ul、li、applet、center、dir、menu、および pre のみです。

**1** "SpryEffects.js" ファイルを Web ページにリンクするには、次のコードをドキュメントの先頭に追加します。

```
<head>
. . .
<script src="../../includes/SpryEffects.js" type="text/javascript" ></script>
</head>
```

**注意:** 正確なファイルパスは、"SpryEffects.js" ファイルの格納場所によって異なります。

"SpryEffects.js" ファイルは、Adobe Labs からダウンロードした "Spry" フォルダの "includes" フォルダにあります。123 ページの「ファイルの準備」を参照してください。

2 ターゲットエレメントに、一意の ID が指定されていることを確認します。ユーザーがページで効果が発生させるインタラクティブな操作を行うと、このターゲットエレメントが変化します。

```
<div id="blindup1">
  <h4>HEADER</h4>
  <p class="sampleText"> Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed
    diam nonumy eirmod tempor invidunt ut labore et dolore magna</p>
</div>
```

3 効果を作成するには、ユーザーがページでインタラクティブな操作を行ったときに効果が発生させる JavaScript イベントを追加します。たとえば、ユーザーが文をクリックしたときに別の段落をブラインドアップするには、次のイベントをその文の p タグに追加します。

```
<p><a onclick="Spry.Effect.DoBlind('blindup1', {duration: 1000, from: '100%', to: '0%', toggle: true});
return false;" href="#" >Click here to blind up from 100% to 0%</a></p>
```

JavaScript メソッドの最初のパラメータは、常にターゲットエレメントの ID (上の例では 'blindup1') です。

コード全体を次に示します。

```
<head>
. . .
<script src="../../includes/SpryEffects.js" type="text/javascript" > </script>
<style type="text/css">
#blindup1{
  background-color: #CCCCCC;
  height: 200px;
  width: 300px;
  overflow: hidden;
}
</style>
</head>
<body>
<p><a onclick="Spry.Effect.DoBlind('blindup1', {duration: 1000, from: '100%', to: '0%', toggle: true});
return false;" href="#" >Click here to blind up from 100% to 0%</a></p>
<div id="blindup1">
  <h4>HEADER</h4>
  <p class="sampleText"> Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed
    diam nonumy eirmod tempor invidunt ut labore et dolore magna</p>
</div>
</body>
```

### ブラインドアップ/ブラインドダウン効果のオプション

次の表は、ブラインドアップ/ブラインドダウン効果で使用できるオプションのリストです。

| オプション      | 説明  |
|------------|---|
| duration   | 効果の継続時間をミリ秒単位で指定します。デフォルト値は 1000 です。  |
| from       | 開始サイズを % またはピクセル単位で指定します。デフォルト値は 100% です。   |
| to         | 終了サイズを % またはピクセル単位で指定します。デフォルト値は 0% です。   |
| toggle     | 切り替え効果を生成します。デフォルト値は false です。  |
| transition | 移行のタイプとして linear または sinusoidal を指定します。linear では、移行が継続する間、移行速度が一定になります。sinusoidal では、効果がゆるやかに開始され、加速した後、再び減速して終了します。デフォルトは sinusoidal です。 |
| setup      | 効果の開始前に呼び出す関数を定義できます。たとえば、 <code>setup:function (element, effect) { /* ... */ }</code> のように指定します。   |
| finish     | 効果の終了後に呼び出す関数を定義できます。たとえば、 <code>finish:function (element, effect) { /* ... */ }</code> のように指定します。  |

サンプルコード:

```
Spry.Effect.DoBlind('targetID', {duration: 1000, from: '100%', to: '0%'});
```

## スライド効果の適用

スライド効果は、ターゲットエレメントを上下（または左右）に移動します。この効果はブラインド効果に似ていますが、コンテンツが同じ位置に留まるのではなく上下（または左右）に移動します。

この効果を適用できる HTML エレメントは、blockquote、dd、div、form、center、table、span、input、textarea、select、および image のみです。

**1** "SpryEffects.js" ファイルを Web ページにリンクするには、次のコードをドキュメントの先頭に追加します。

```
<head>
. . .
<script src="../../includes/SpryEffects.js" type="text/javascript" ></script>
</head>
```

**注意：**正確なファイルパスは、"SpryEffects.js" ファイルの格納場所によって異なります。

"SpryEffects.js" ファイルは、Adobe Labs からダウンロードした "Spry" フォルダの "includes" フォルダにあります。123 ページの「ファイルの準備」を参照してください。

**2** ターゲットエレメントが、一意の ID を持つ div で囲まれていることを確認します。ユーザーがページで効果を発生させるインタラクティブな操作を行うと、このターゲットエレメントが変化します。

```
<div id="slide1">
  <div> Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed
  diam nonumy eirmod tempor invidunt ut labore et dolore magna
  aliquyam erat, sed diam voluptua.</div>
</div>
```

**3** 効果を作成するには、ユーザーがページでインタラクティブな操作を行ったときに効果を発生させる JavaScript イベントを追加します。たとえば、ユーザーが文をクリックしたときに別の段落をスライドアップするには、次のイベントをその文の p タグに追加します。

```
<p><a onclick="Spry.Effect.DoSlide('slide1',{duration:1000,from:'100%', to:'20%',toggle:true}); return
false;" href="#">Click here to slide the paragraph up from 100% to 20%</a></p>
```

JavaScript メソッドの最初のパラメータは、常にターゲットエレメントの ID（上の例では、'slide1'）です。

コード全体を次に示します。

```
<head>
. . .
<script src="../../includes/SpryEffects.js" type="text/javascript" > </script>
</head>
<body>
<p><a onclick="Spry.Effect.DoSlide('slide1',{duration:1000,from:'100%', to:'20%',toggle:true}); return
false;" href="#"> Click here to slide the paragraph up from 100% to 20%</a></p>
<div id="slide1">
  <div> Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed
  diam nonumy eirmod tempor invidunt ut labore et dolore magna
  aliquyam erat, sed diam voluptua.</div>
</div>
</body>
```

## スライド効果のオプション

次の表は、スライド効果で使用できるオプションのリストです。

| オプション    | 説明  |
|----------|---|
| duration | 効果の継続時間をミリ秒単位で指定します。デフォルト値は 2000 です。        |
| from     | 開始サイズを % またはピクセル単位で指定します。デフォルト値は '100%' です。 |
| to       | 終了サイズを % またはピクセル単位で指定します。デフォルト値は '0%' です。   |
| toggle   | 切り替え効果を生成します。デフォルト値は false です。              |

| オプション      | 説明  |
|------------|---|
| transition | 移行のタイプとして linear または sinusoidal を指定します。linear では、移行が継続する間、移行速度が一定になります。sinusoidal では、効果がゆるやかに開始され、加速した後、再び減速して終了します。デフォルトは sinusoidal です。 |
| horizontal | true に設定すると、コンテンツが垂直ではなく水平にスライドします。デフォルト値は false です。  |
| setup      | 効果の開始前に呼び出す関数を定義できます。たとえば、 <code>setup:function (element, effect) { /* ... */ }</code> のように指定します。   |
| finish     | 効果の終了後に呼び出す関数を定義できます。たとえば、 <code>finish:function (element, effect) { /* ... */ }</code> のように指定します。  |

サンプルコード：

```
Spry.Effect.DoSlide('targetID',{duration: 1000,from: '100%',to: '0%'});
```

## 拡張効果の適用

拡張効果は、エレメントのサイズを拡張または縮小します。エレメントの中央に向かって小さくなるか、中央から広がるモーションです。

この効果を適用できる HTML オブジェクトは、address、dd、div、dl、dt、form、p、ol、ul、applet、center、dir、menu、img、および pre のみです。

**1** "SpryEffects.js" ファイルを Web ページにリンクするには、次のコードをドキュメントの先頭に追加します。

```
<head>
. . .
<script src="../../includes/SpryEffects.js" type="text/javascript" ></script>
</head>
```

**注意：**正確なファイルパスは、"SpryEffects.js" ファイルの格納場所によって異なります。

"SpryEffects.js" ファイルは、Adobe Labs からダウンロードした "Spry" フォルダの "includes" フォルダにあります。123 ページの「ファイルの準備」を参照してください。

**2** ターゲットエレメントに、一意の ID が指定されていることを確認します。ユーザーがページで効果を発生させるインタラクティブな操作を行うと、このターゲットエレメントが変化します。

```
<div class="demoDiv" id="shrink1">
<p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum.</p>
</div>
```

**3** 効果を作成するには、ユーザーがページでインタラクティブな操作を行ったときに効果を発生させる JavaScript イベントを追加します。たとえば、ユーザーが文をクリックしたときに別の段落を縮小するには、次のイベントをその文の p タグに追加します。

```
<p><a onclick="Spry.Effect.DoGrow('shrink1',{duration:700, from:'100%', to:'20%',toggle: true}); return false;" href="#">Click here to shrink the paragraph from 100% to 20%.</a></p>
```

JavaScript メソッドの最初のパラメータは、常にターゲットエレメントの ID (上の例では、'shrink1') です。

コード全体を次に示します。

```

<head>
. . .
<script src="../../includes/SpryEffects.js" type="text/javascript" ></script>
</head>
<body>
<p><a onclick="Spry.Effect.DoGrow('shrink1',{duration:700, from:'100%', to:'20%',toggle: true}); return
false;" href="#">Click here to shrink the paragraph from 100% to 20%.</a></p>
<div class="demoDiv" id="shrink1">
<p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut
labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et
ea rebum.</p>
</div>
</body>

```

### 拡張効果のオプション

次の表は、拡張効果で使用できるオプションのリストです。

| オプション      | 説明  |
|------------|---|
| duration   | 効果の継続時間をミリ秒単位で指定します。デフォルト値は 500 です。   |
| from       | 開始サイズを % またはピクセル単位で指定します。デフォルト値は 0% です。   |
| to         | 終了サイズを % またはピクセル単位で指定します。デフォルト値は 100% です。   |
| toggle     | 切り替え効果を生成します。デフォルト値は false です。  |
| growCenter | エレメントの拡張および縮小の方向を指定します。デフォルト値は、true (中央から拡張および縮小) です。false に設定すると、エレメントは、左上隅から拡張および縮小します。   |
| transition | 移行のタイプとして linear または sinusoidal を指定します。linear では、移行が継続する間、移行速度が一定になります。sinusoidal では、効果がゆるやかに開始され、加速した後、再び減速して終了します。デフォルトは sinusoidal です。 |
| setup      | 効果の開始前に呼び出す関数を定義できます。たとえば、setup:function (element, effect) { /* ... */ } のように指定します。   |
| finish     | 効果の終了後に呼び出す関数を定義できます。たとえば、finish:function (element, effect) { /* ... */ } のように指定します。  |

サンプルコード:

```
Spry.Effect.DoGrow('targetID',{duration: 1000,from: '0%', to: '100%'});
```

### スキッシュ効果の適用

スキッシュ効果は、ターゲットエレメントをページの左上隅に消します。スキッシュ効果は、拡張効果の growCenter オプションを false に設定した場合と同じ効果を生じます。

この効果を適用できる HTML エレメントは、address、dd、div、dl、dt、form、img、p、ol、ul、applet、center、dir、menu、および pre のみです。

**1** "SpryEffects.js" ファイルを Web ページにリンクするには、次のコードをドキュメントの先頭に追加します。

```

<head>
. . .
<script src="../../includes/SpryEffects.js" type="text/javascript" ></script>
</head>

```

**注意:** 正確なファイルパスは、"SpryEffects.js" ファイルの格納場所によって異なります。

"SpryEffects.js" ファイルは、Adobe Labs からダウンロードした "Spry" フォルダの "includes" フォルダにあります。123 ページの「ファイルの準備」を参照してください。

**2** ターゲットエレメントに、一意の ID が指定されていることを確認します。ユーザーがページで効果が発生させるインタラクティブな操作を行うと、このターゲットエレメントが変化します。

```
<div id="squish1"><p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquet</p></div>
```

**3** 効果を作成するには、ユーザーがページでインタラクティブな操作を行ったときに効果が発生させる JavaScript イベントを追加します。たとえば、ユーザーが文をクリックしたときに別の段落をスキッシュするには、次のイベントをその文の `p` タグに追加します。

```
<p><a onclick="Spry.Effect.DoSquish('squish1'); return false;" href="#">Click here to squish the paragraph.</a></p>
```

JavaScript メソッドの最初のパラメータは、常にターゲットエレメントの ID (上の例では、'squish1') です。

コード全体を次に示します。

```
<head>
. . .
<script src="../../includes/SpryEffects.js" type="text/javascript" ></script>
</head>
<body>
<p><a onclick="Spry.Effect.DoSquish('squish1'); return false;" href="#">Click here to squish the paragraph.</a></p>
<div id="squish1"><p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquet</p></div>
</body>
```

### スキッシュ効果のオプション

次の表は、任意で利用できるオプションのリストです。

| オプション    | 説明   |
|----------|--|
| duration | 効果の継続時間をミリ秒単位で指定します。デフォルト値は 1000 です。   |
| toggle   | 切り替え効果を生成します。デフォルト値は false です。   |
| setup    | 効果の開始前に呼び出す関数を定義できます。たとえば、 <code>setup:function (element, effect) { /* ... */ }</code> のように指定します。  |
| finish   | 効果の終了後に呼び出す関数を定義できます。たとえば、 <code>finish:function (element, effect) { /* ... */ }</code> のように指定します。 |

サンプルコード：

```
Spry.Effect.DoSquish('targetID', {duration: 1000});
```

### シェイク効果の適用

シェイク効果は、ターゲットエレメントが左右に 20 ピクセルずつすばやく揺れているように見せます。

この効果を適用できる HTML エレメントは、address、blockquote、dd、div、dl、dt、fieldset、form、h1、h2、h3、h4、h5、h6、iframe、img、object、p、ol、ul、li、applet、dir、hr、menu、pre、および table のみです。

**1** "SpryEffects.js" ファイルを Web ページにリンクするには、次のコードをドキュメントの先頭に追加します。

```
<head>
. . .
<script src="../../includes/SpryEffects.js" type="text/javascript" ></script>
</head>
```

**注意：**正確なファイルパスは、"SpryEffects.js" ファイルの格納場所によって異なります。

"SpryEffects.js" ファイルは、Adobe Labs からダウンロードした "Spry" フォルダの "includes" フォルダにあります。123 ページの「ファイルの準備」を参照してください。

**2** ターゲットエレメントに、一意の ID が指定されていることを確認します。ユーザーがページで効果が発生させるインタラクティブな操作を行うと、このターゲットエレメントが変化します。

```
<div id="shake1">Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquet amet.</div>
```

**3** 効果を作成するには、ユーザーがページでインタラクティブな操作を行ったときに効果が発生させる JavaScript イベントを追加します。たとえば、ユーザーが文をクリックしたときに別の段落をシェイクするには、次のイベントをその文の p タグに追加します。

```
<p><a onclick="Spry.Effect.DoShake('shake1'); return false;" href="#">Shake it!</a></p>
```

JavaScript メソッドの最初のパラメータは、常にターゲットエレメントの ID (上の例では、'shake1') です。

コード全体を次に示します。

```
<head>
. . .
<script src="../../includes/SpryEffects.js" type="text/javascript" ></script>
</head>
<body>
<p><a onclick="Spry.Effect.DoShake('shake1'); return false;" href="#">Shake it!</a></p>
<div id="shake1">Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor
invidunt ut labore et dolore magna aliquit amet.</div>
</body>
```

### シェイク効果のオプション

次の表は、シェイク効果で利用できるオプションのリストです。

| オプション    | 説明   |
|----------|--|
| duration | Spry 1.4 では、500 ミリ秒に固定されています。Spry 1.5 以降でのみ編集可能です。   |
| setup    | 効果の開始前に呼び出す関数を定義できます。たとえば、 <code>setup:function (element, effect) { /* ... */ }</code> のように指定します。  |
| finish   | 効果の終了後に呼び出す関数を定義できます。たとえば、 <code>finish:function (element, effect) { /* ... */ }</code> のように指定します。 |

サンプルコード：

```
Spry.Effect.DoShake('targetID');
```

# 索引

## D

Dreamweaver CS3 1

## J

JavaScript の適用範囲縮小 2

## S

Spry

Dreamweaver CS3 1

Widget、説明 2

フレームワーク、説明 1

## X

XML データセット 80

概要 80

高度な例 82

作成 97

ユーザーによるデータのソート 101

## あ

アクセシビリティ 2, 95

アコーディオン Widget

カスタマイズ 10

キーボードによる操作の有効化 10

最初に開くパネルの設定 10

説明 4

挿入 7

パネルの削除 9

パネルの追加 9

## お

折りたたみパネル Widget

カスタマイズ 18

キーボードによる操作の有効化 17

最初に開くパネルの設定 17

説明 13

挿入 15

## か

拡張効果 128

## こ

効果

説明 122

効果ライブラリ

説明 122

## し

シェイク効果 130

## す

スキッシュ効果 129

スライド効果 127

## せ

選択検査 Widget

カスタマイズ 72

検査を実行するタイミングの指定 71

説明 66

挿入 68

必須状態の変更 71

無効な値の指定 71

## た

タブ付きのパネル Widget

カスタマイズ 27

キーボードによる操作の有効化 26

最初に開くパネルの設定 27

説明 21

挿入 24

パネルの削除 26

パネルの追加 26

## ち

チェックボックス検査 Widget

カスタマイズ 78

検査を実行するタイミングの指定 77

最小および最大選択オプション数の指定 78

説明 73

挿入 75

必須状態の変更 78

## て

データ

オブザーバー通知 110

キャッシュのオフ 107

現在の行、設定と変更 109

更新 110

参照、非表示 119

取得 106, 107

ソート 108

重複行の削除 109

フィルタ処理 109

**テキストフィールド検査 Widget**

- カスタマイズ 54
- 検査の種類とフォーマットの指定 44
- 検査を実行するタイミングの指定 53
- 最小および最大文字数の指定 53
- 最小値および最大値の指定 53
- 説明 39
- 挿入 42
- 必須状態の変更 54
- ヒントの作成 54
- 無効な文字のブロック 54

**テキスト領域検査 Widget**

- カスタマイズ 64
- 検査を実行するタイミングの指定 62
- 説明 58
- 挿入 60
- 必須状態の変更 64
- ヒントの作成 64
- 文字カウンタの追加 63
- 余分な文字のブロック 64

**と****動的領域**

- オブザーバー通知 117
- 概要 86
- 作成 99
- 条件構造 115
- データの表示 99
- 領域の状態 116
- ループ構造 112

**は**

- ハイライト効果 123

**ひ**

- ビヘイビア属性 120

**ふ**

- フェード効果 124
- ブラインドアップ/ブラインドダウン効果 125

**ま****マスター領域と詳細領域**

- 概要 88
- 作成 102, 104
- 複数のデータセットに依存 91

**め****メニューバー Widget**

- カスタマイズ 37
- 説明 29
- 挿入 33
- 方向の変更 36