

## Caché Server Pages クイックスタート・チュートリアル

Caché Server Pages クイックスタート・チュートリアルへようこそ

Caché Server Page(CSP)は、高性能なデータ駆動型の Web アプリケーションを迅速に開発および配置するためのプラットフォームです。このプラットフォームには、さまざまなプログラミングと配置モデルがあります。このチュートリアルでは、CSPを使用した Web アプリケーションの開発と配置の概要を実践に基づき短時間で説明します。

このチュートリアルは2つの章から構成されています。チュートリアルの第2章を正しく実行するには、第1章で示す情報を理解する必要があります。

1. "[第1章:CSPとは](#)"では、CSPページのライフサイクル、構成、[%CSP.Page](#)クラス、CSPマークアップ要素(式、スクリプト、タグなど)といったCSPの基本的なトピックについて学習します。
2. "[第2章:CSPプログラミングの基本](#)"では、オブジェクト・バインディング、クエリの実行、状態の管理、ハイパーイベントの使用、CachéスタジオのWebフォーム・ウィザードの使用といったCSPプログラミングのトピックについて学習します。

### Note:

このチュートリアルは、Caché ObjectScriptに関する実践的な知識を備えていることを前提としています。Caché ObjectScriptの詳細は、Cachéドキュメント:[\[Caché チュートリアル\]-\[Caché ObjectScript チュートリアル\]](#)または[\[Cachéクイックスタート・チュートリアル\]](#)を参照してください。CachéおよびCSPを使用したWebアプリケーションの構築に関する説明の詳細は、Cachéドキュメント:[\[Caché チュートリアル\]-\[Caché Webアプリケーション・チュートリアル\]](#)を参照してください。チュートリアルには、例と解答ファイルが数多く含まれています。これらのインストールの手順については、[\[Appendix A: チュートリアル・ファイルのインストール\]](#) を参照してください。

## はじめに

このチュートリアル第 1 章では、CSP の基本的なトピックについて説明します。この章には、演習や例を収めたファイルがいくつか用意されています。この章の学習を始める前に、これらのファイルをインストールする必要があります。手順については、下記のメモを参照してください。

この章の学習を終えると、以下のことを実行できるようになります。

- CSP プラットフォームの主要コンポーネント名を一覧にして、その役割を説明する。
- CSP ページへの Web ブラウザのリクエストを実行するプロセスの基本手順を説明する。
- CSP の基本コンポーネントの構成設定を説明する。
- URL の各部分が CSP ページへのマッピングを決定する方法を説明する。
- CSP ページを開発するための 2 つの異なる方法について説明する。
- CSP ページのライフサイクルにおける 3 つの主なフェーズについて説明する。
- Caché スタジオを使用して簡単な CSP ページを作成し、Web ブラウザにページを表示する。
- 5 種類の主な CSP マークアップ要素をそれぞれ Caché コードに変換する CSP コンパイラについて説明する。
- 5 種類の主な CSP マークアップ要素をそれぞれ CSP ページに追加する。

### Note:

この章に付随するファイルのインストール手順については、[Appendix A: チュートリアル・ファイルのインストール] を参照してください。

## CSP とは

Caché Server Page(CSP)は、動的な Web アプリケーションを開発および配置するためのプラットフォームです。CSP プラットフォームの主なコンポーネントは以下のとおりです。

- Web サーバ — クライアント(通常は Web ブラウザ)からの HTTP 要求を処理して、CSP コンテンツの要求を CSP ゲートウェイに転送するアプリケーション。CSP ゲートウェイから受信したコンテンツをクライアントに返します。Caché では、Apache および Microsoft Internet Information Server(IIS)といったサードパーティの Web サーバを数多くサポートしています。
- CSP ゲートウェイ — Web サーバによってインストールおよびロードされる共有ライブラリまたはダイナミック・リンク・ライブラリ(DLL)。ゲートウェイは、CSP コンテンツの要求を Web サーバから CSP サーバに転送し、CSP サーバで生成した CSP コンテンツを Web サーバに返します。
- CSP コンパイラ — CSP サーバ上に配置される Caché クラスとプログラムのセット。コンパイラは、CSP マークアップ・ページを Caché ページ・クラスに変換して、ページ・クラスをコンパイルします。
- CSP サーバ — CSP ゲートウェイから受信する CSP 要求を処理するアプリケーション。CSP コンパイラを使用して、CSP コードを変換およびコンパイルします。CSP クラスをインスタンス化してから、その **onPage** メソッドを呼び出して、CSP コンテンツを生成します。
- **%CSP** API — 開発者がCSPクラスのコード化に使用できるクラスのパッケージ。このパッケージには、すべてのCSPクラスのスーパークラスである[%CSP.Page](#)が含まれません。
- CSP マークアップ言語 — 開発者が HTML マークアップと組み合わせて CSP ページを作成できるタグと指示文のセット。

### Note:

標準のCachéインストールにはWebサーバが含まれます。既定では、このWeb サーバは最初に使用可能な 57772 以上のポート番号を使用します。このWeb サーバの構成と、Cachéで使用する他のWebサーバの構成の詳細は、Cachéドキュメント: [\[Caché開発ガイド\]](#)-[\[Caché Server Pages \(CSP\)の使用法\]](#)-[\[CSP アーキテクチャ\]](#)を参照してください。

Web サーバとCSP ゲートウェイは、同じ物理マシン上に置く必要があります。ただし、それ以外のコンポーネントは異なるマシン上でも同じマシン上でもかまいません。

## 要求の処理

一般的な CSP アプリケーションは、クライアント要求に 응답して動的な Web ページを生成します。これらの Web ページには、データ、HTML、および JavaScript が混在します。クライアント(ほとんどの場合 Web ブラウザ)が CSP アプリケーションの CSP ページを要求した後の一連のイベントは以下ようになります。

1. Web サーバは、ページに対する要求(特に HTTP 要求)を受信します。要求されたページのファイル拡張子が .csp または .cls の場合(<http://.../.../mypage.csp> など)、Web サーバはその要求を CSP ゲートウェイに転送します。
2. CSP ゲートウェイは、その CSP ページに対する要求を再パッケージ化して、適切な CSP サーバに転送します。
3. CSP サーバは要求をデコードして、要求された CSP アプリケーションとクラス (**MyPage.cls**)を決定します。さらに、クラスのインスタンスを作成し、その **onPage** メソッドを呼び出します。
4. CSP サーバは、**onPage** メソッドの出力を HTTP 応答にまとめ、
5. その HTTP 応答を CSP ゲートウェイに送信します。CSP ゲートウェイは Web サーバに HTTP 応答を送信し、Web サーバは最初の要求を送信したクライアントに HTTP 応答を返します。



## 基本的な構成

Web サーバ、CSP ゲートウェイ、CSP サーバはすべて、連携して動作するように構成する必要があります。

### CSP 構成

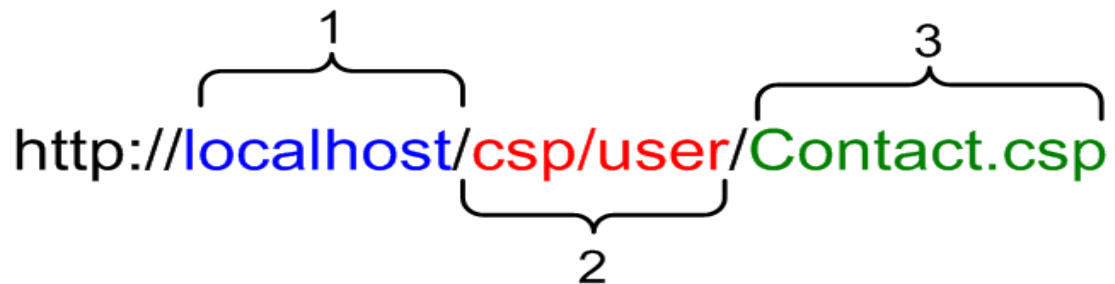
コンポーネント	設定
Web サーバ	<ul style="list-style-type: none"> <li>仮想ディレクトリ — Web サーバが物理ディレクトリにマップするディレクトリ名。CSP 以外のコンテンツを物理ディレクトリに配置します。</li> <li>アクセス・コントロール — ユーザを認証し、リソースへのアクセスを承認するための Web サーバのメカニズム。</li> </ul>
CSP ゲートウェイ	<ul style="list-style-type: none"> <li>アプリケーション・パス — 1 つの CSP アプリケーションを構成する CSP ページの共通アドレスを提供する URL パスの一部。</li> <li>サーバ・パス — Caché サーバのアドレスとポート番号。</li> </ul> <p>メモ: ブラウザを介して CSP ゲートウェイにアクセスします。Windows の IIS を使用する場合、アドレス <code>http://localhost/csp/bin/cspmssys.dll</code> を使用します。Windows の Apache v2 を使用する場合、アドレス <code>http://localhost/csp/bin/systems/module.cwx</code> を使用します。</p>
CSP サーバ	<ul style="list-style-type: none"> <li>物理パス — CSP ファイルを格納するディレクトリ。このパスは、<code>&lt;cache-install&gt;/csp</code> を基準とする相対パスです。</li> <li>ネームスペース — アプリケーションの Caché ネームスペース。</li> </ul> <p>その他にも、Cookie、タイムアウト、カスタム・エラー・ページなどを構成できるサーバ関連の設定がいくつかあります。</p>

#### Note:

標準の Caché インストールでは、CSP ページの追加構成を USER ネームスペースに追加する必要はありません。

この 3 つのコンポーネントの構成についてさらに学習するには、Cachéドキュメント: [\[Caché開発ガイド\]-\[Caché Server Pages \(CSP\)の使用法\]-\[CSP アーキテクチャ\]](#)を参照してください。

URL の各部分が、CSP ページへのマッピングを決定します。



1. Webサーバ。Webサーバの名前またはIPアドレス (127.0.0.1:57772 など)です。
2. 仮想ディレクトリおよびアプリケーション・パス。最初のレベルは仮想ディレクトリです。Webサーバで構成します。残りのレベルはアプリケーション・パスです。CSPサーバの構成時にこのパスを構成します。
3. CSP ページ名。要求された CSP ページの名前です。この名前には、拡張子.cspまたは.cls を付ける必要があります。

## CSP 開発の方法

開発者は、CSP ページを作成して配置します。これらのページは、アプリケーションがクライアントに返すコンテンツを生成します。CSP ページを開発するには、以下の 2 つの方法があります。

1. Caché ページ・クラスを直接コーディングできます。CSP ページ・クラスは、[%CSP.Page](#) を拡張する必要があります。このページ・クラスを最初にコーディングするには、[%CSP.Page](#) から継承したメソッドをオーバーライドします。すべての Caché クラスと同様、CSP ページ・クラスを、拡張子 .cls のファイルに含める必要があります。
2. HTML や CSP マークアップと、Caché ObjectScript または Caché Basic スクリプトが混在するマークアップ・ページを作成できます。CSP コンパイラは、このマークアップ・ページを、対応する Caché ページ・クラスに自動的に変換してから、コンパイルを行います。このマークアップ・ページは拡張子 .csp のファイルに含めてください。

2 番目の方法は、一般的に最も便利なものです。通常、Caché のコーディングが比較的少なくて済むこの方法は、Caché の豊富なプログラミング経験がない開発者にとって特に有用です。このチュートリアルでは、CSP 開発の 2 番目の方法について説明します。

## CSP ページのライフサイクル

CSP ページのライフサイクルには、主に以下の 3 つのフェーズがあります。

1. 設計時間 — 開発者は、HTML マークアップおよび CSP 要素（ページ指示文、Caché データ式、CSP タグ、CSP スクリプトなど）を使用して、拡張子 .csp のマークアップ・ファイルを作成します。最終的に、CSP コンパイラがこのマークアップ・ファイルを Caché クラスに変換します。あるいは、開発者が Caché クラスを直接コード化します。
2. コンパイル時間 — CSP コンパイラは、まず設計時間で作成したマークアップ・ファイルを Caché クラスに変換します。変換時に、CSP コンパイラは、.csp ファイル内の CSP 要素のタイプと位置に従って、生成されるクラスに CSP 要素を配置し、クラスをコンパイルします。
3. 実行時間 — Caché サーバはコンパイル時間で作成したクラスのインスタンスを作成し、その `onPage` メソッドを呼び出します。CSP ページ・クラスは、[%CSP.Page](#) から `onPage` を継承します。

## CSP 要素とマークアップ言語

以下のテーブルでは、設計時間に CSP マークアップ・ファイルに追加できる各種 CSP 要素とマークアップの一部について説明します。

### CSP 式

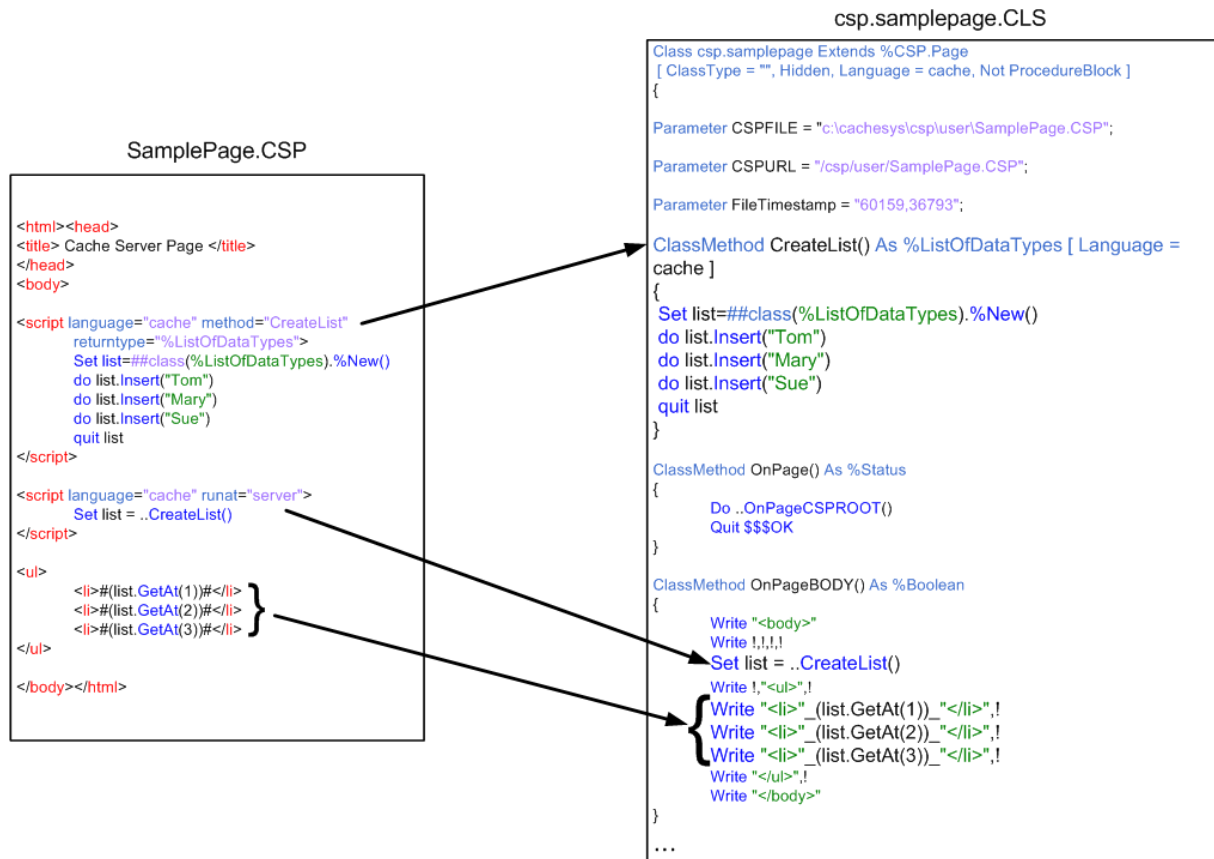
要素	説明
<code>#{Expr}#</code>	実行時式。Expr は任意の Caché Basic または ObjectScript 式です。Caché サーバは、ページの実行時に式を評価します。評価の結果は、ページの出力に挿入されます。
<code>##(Expr)##</code>	コンパイル時の式。Expr は任意の ObjectScript 式です (Caché Basic 式にはできないことに注意してください)。Caché コンパイラは、CSP ページ・クラスのコンパイル時に式を評価します。実行時に、ページはその出力に結果を挿入します。

### CSP スクリプトとマークアップ言語

要素	説明
<pre>&lt;script language=("CACHE" "BASIC") runat="Server"&gt; ObjectScript または Basic Code &lt;/script&gt; &lt;script language=("CACHE" "BASIC") method="" arguments="" returntype=""&gt; ObjectScript または Basic Code &lt;/script&gt;</pre>	<p>CSP スクリプト。ページの <code>&lt;script&gt;&lt;/script&gt;</code> タグの間に複数行のコードを挿入することができます。Caché サーバは実行時にコードを実行し、その結果はページの出力に挿入されます。</p>
<pre>&lt;CSP:XXX&gt;</pre>	<p>CSP メソッド。この構文を使用して CSP 文書内にメソッドを定義することができます。メソッド名、引数リスト、および返りタイプを指定することができます。例えば、CSP 式または CSP スクリプトにより、ページの他の場所でメソッドを呼び出すことができます。コードは、Caché Basic または ObjectScript のどちらでもかまいません。</p> <p>CSP マークアップ言語は、HTML ページの生成時または CSP ページのコンパイル時にコードを実行するタグのセットで構成されます。これらのタグは、例えば、ページ実行のフローを制御し、データへのアクセスを提供するために使用することができます。</p>

## CSP ページから Caché クラスへの変換

CSP サーバは、すべての CSP ページを Caché クラスに自動的に変換します。次の図は、左側に CSP ファイル、右側に Caché クラスに変換したものを示しています。



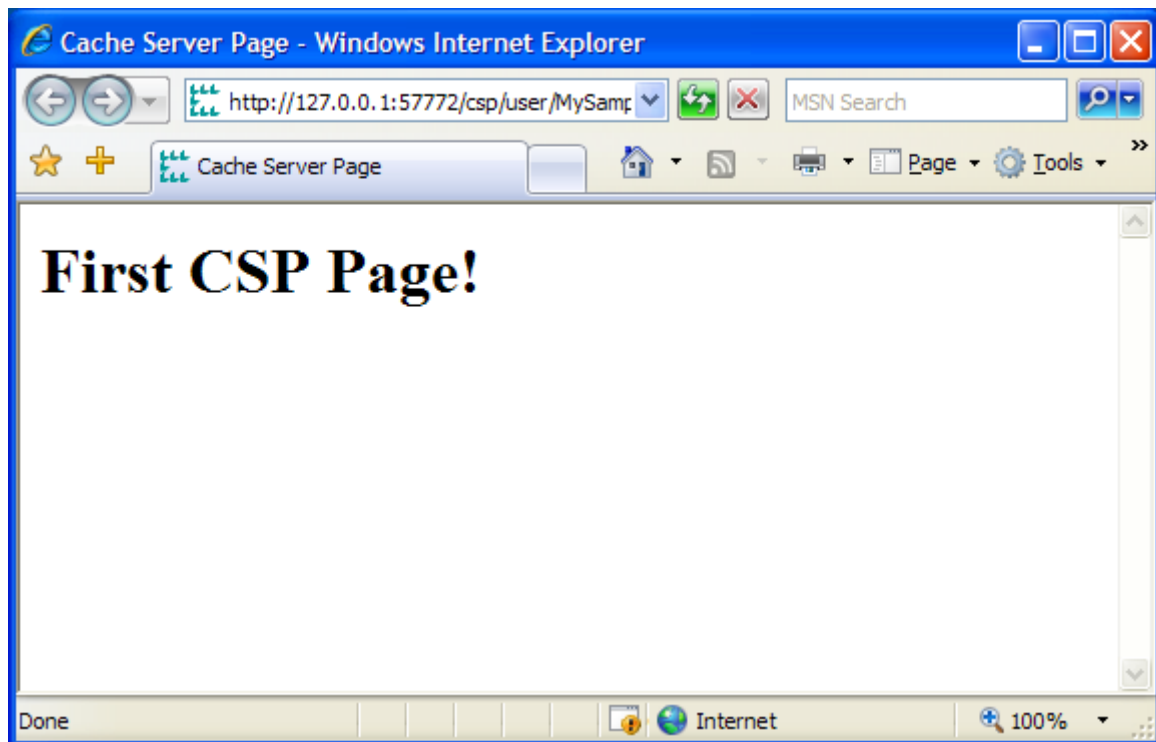
## CSP サンプル・ページ

Caché スタジオの CSP ウィザードを使用して、MySamplePage.CSP という名前の CSP ページを作成するには、以下の手順を実行します。このチュートリアルの第 1 章の後半では、MySamplePage.CSP を使用して、さまざまな CSP 要素と機能を実際に試してみます。

1. USER ネームスペースでスタジオを開きます。スタジオが USER で開かない場合、**[ファイル]**→**[ネームスペース変更]**をクリックして USER に接続します。
2. スタジオのメニュー・バーから**[ファイル]**→**[新規作成]**をクリックします。
3. **[新規作成]**ウィザードの**[CSP ファイル]**タブをクリックします。
4. **[CSP ページ]**アイコンをクリックします。スタジオに新しい CSP マークアップ・ページが表示されます。このページには、`<html></html>`、`<head></head>`、`<body></body>`タグなどの HTML ページの基本構造が含まれます。
5. ページに現在ある`<body></body>`タグの間にテキストと HTML フォーマット・タグを追加します。ウィザードでは、この場所に「My Page Body」というテキストが自動的に配置されます。出力に表示されないようにこのテキストを削除します。

```
...
<body>
<h1>First CSP Page!</h1>
</body>
...
```

6. ファイルに名前を付けて保存するには、**[ファイル]**→**[名前を付けて保存]**をクリックします。ファイルの名前を MySamplePage.CSP にします。必ず CSP/USER にファイルを保存してください。これは、スタジオが USER ネームスペースに接続している場合の既定です。
7. **[表示]**→**[ブラウザで表示]**をクリックすると、Web ブラウザに MySamplePage.CSP が表示されます。



**Note:**

CSPコンパイラは、CSP マークアップ・ページの<body></body>タグの間にあるすべてのテキストとHTMLを、対応するCSP ページ・クラスの `onPageBody` メソッドの `Write` 文に配置します。

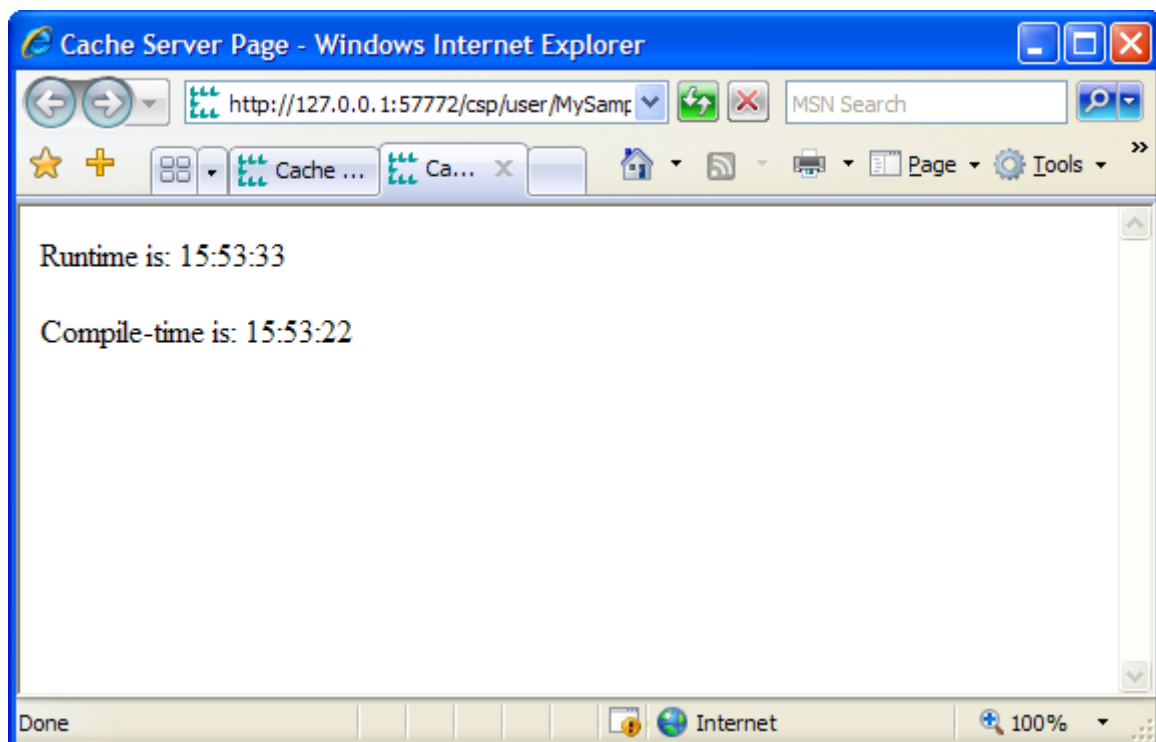
## CSP 要素:式

CSP 式を使用すると、ObjectScript(または Caché Basic)式を CSP ページに挿入できます。これらの式は実行時(#(expr)#)またはコンパイル時(##(expr)##) に評価され、結果はクライアントに返される HTML に追加されます。

以下のテキストと CSP 要素を、MySamplePage.CSP の<body></body>タグの間に追加します。<body>と</body>との間にあるその他のコードはすべて削除できます。

```
<body>
<p>Runtime is: #($zt($p($h,"",2)))#</p>
<p>Compile-time is: ##($zt($p($h,"",2)))##</p>
</body>
```

上記の式が評価されると、どちらも“現在の”時刻を返します。ただし、この 2 つの式はページのライフサイクルの異なる段階で評価されるため、結果として生じる時刻は異なります。最初の式 #(expr)# は実行時式です。これは CSP ページの実行時に評価されます。2 番目の式 ##(expr)## はコンパイル時の式です。これはページ・クラスのコンパイル時に評価されます。コンパイルの後に実行されるため、最初の時刻は 2 番目の時刻よりも必ず遅くなります。まず、[ビルド]→[コンパイル]をクリックしてページをコンパイルします。次に、[表示]→[ブラウザで表示]をクリックすると、Web ブラウザに MySamplePage.CSP が表示されます。最初のタイムスタンプが 2 番目のタイムスタンプよりも遅いことを確認します。



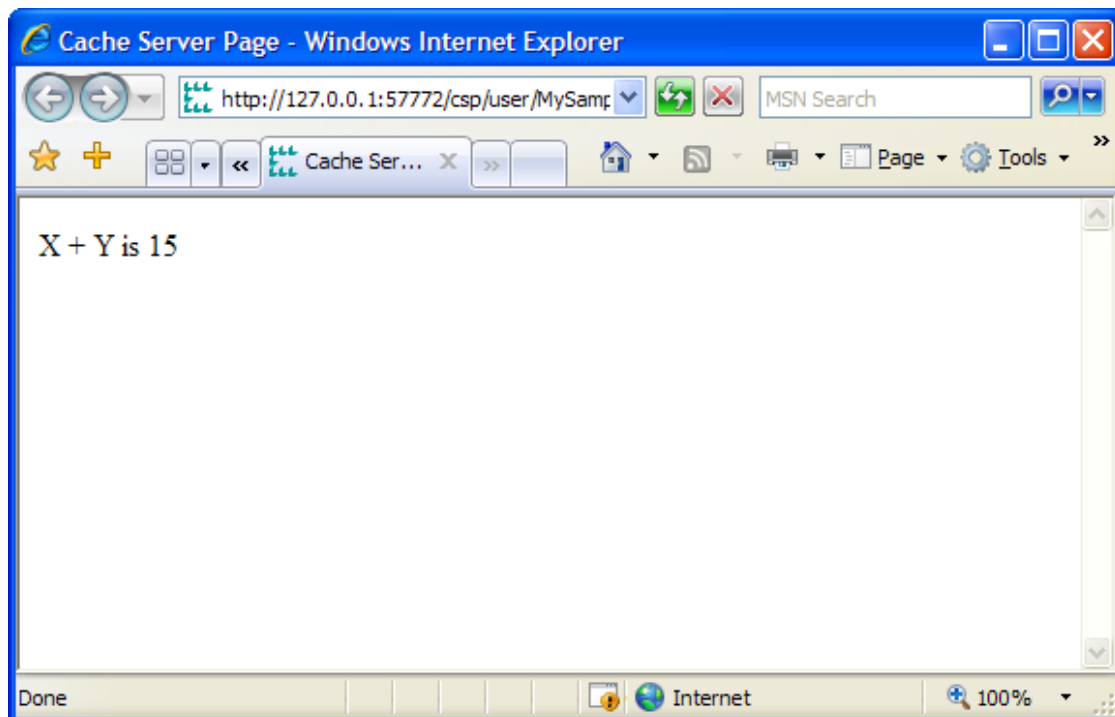
## CSP 要素: Scripts

script タグを使用すると、CSP ページに ObjectScript(または Caché Basic)コード・ブロックを挿入できます。Caché コンパイラは、script タグに含まれるコードを、ページ・クラスの **onPageBody** メソッドに追加します。スクリプトは、**onPageBody** の他のコードと一緒に実行されます。**onPageBody** 内のスクリプトおよび式の配置と、最終的にこれらが実行される順序は、.csp ファイル内のスクリプトおよび式の配置によって決まります。

以下のスクリプトと式を、MySamplePage.CSP の<body>と</body>との間に追加します。<body>と</body>との間にあるその他のコードはすべて削除できます。

```
<body>
<script language="Cache" runat="Server">
  set x = 3+4
</script>
<script language="Cache" runat="Server">
  set y = x+1
</script>
<p>X + Y is #(x + y)#</p>
</body>
```

[表示]→[ブラウザで表示]をクリックすると、Webブラウザに MySamplePage.CSP が表示されます。



## CSP 要素:メソッド

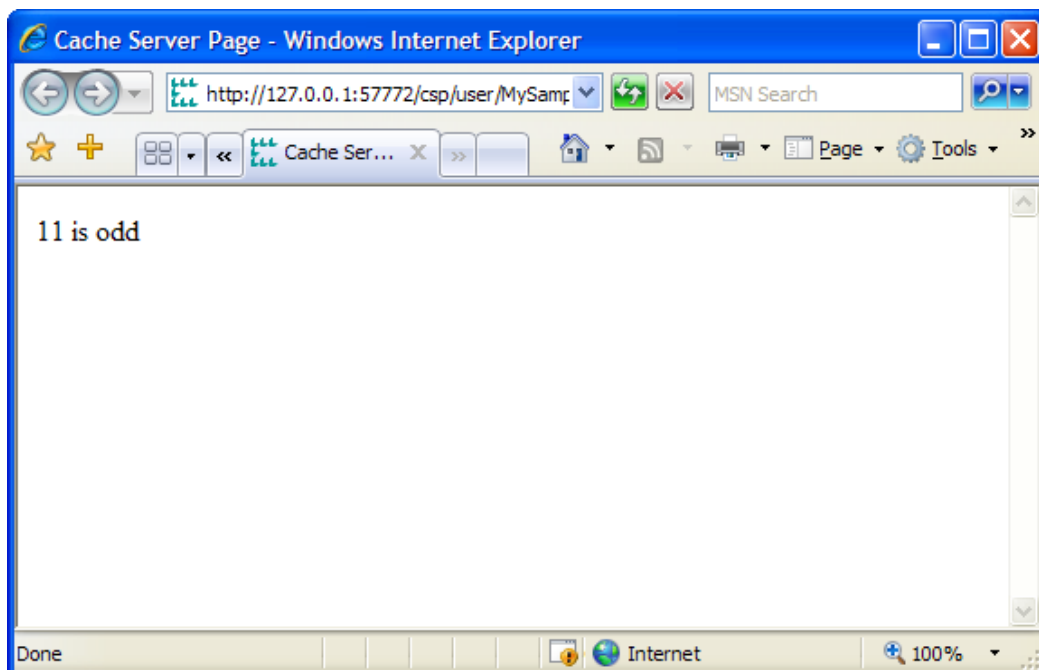
CSP スクリプトを使用すると、CSP ページにメソッドを追加できます。これらのメソッドは、CSP 式などによってページのどの場所からでも呼び出すことができます。script タグを使用してメソッドを定義します。method、arguments、returntype の属性に割り当てられる値が、メソッド名、引数リスト、返りタイプをそれぞれ決定します。タグとタグとの間にあるコードがメソッドの本文になります。

以下のコードを、MySamplePage.CSP の<body>と</body>との間に追加します。

<script></script>タグ間のコードは、偶数の整数が渡される場合は“真”を返し、それ以外の場合は“偽”を返すメソッドを定義します。<p>と</p>との間のコードは、実行時式でメソッドを呼び出します。<body>と</body>との間にあるその他のコードはすべて削除できます。

```
<body>
<p>11 is #(..OddEvenTest(11))#</p> <!--Invoke Method-->
<script language="cache" method="OddEvenTest" arguments="val1:%Integer"
returntype="%String">
  If (val1#2=0) { quit "even" }
  Else {quit "odd"}
</script>
</body>
```

[表示]→[ブラウザで表示]をクリックすると、Web ブラウザに MySamplePage.CSP が表示されます。



**Note:**

その他のクラスのクラス・メソッドは、クラス・メソッド

`##class(PackageName.ClassName).MethodName` を呼び出すための標準の `Caché ObjectScript` 構文を使用して呼び出すことができます。

## CSP マークアップ言語

CSP マークアップ言語は、ページ実行のフロー制御に使用できるタグのセットで構成されます。このタグによって、ObjectScript と Caché Basic 機能の抽象化が可能です。これらのタグを使用すれば、開発者が ObjectScript や Basic に習熟していない場合でも、動的な CSP ページを作成できます。以下で、いくつかのタグについて説明します。

## 一部の CSP タグ

タグ	属性	説明
<code>&lt;csp:include&gt;</code>	<ul style="list-style-type: none"> <li>• <code>page</code>: 挿入先ページの URL。</li> </ul>	あるページのコンテンツを別のページに組み込むために使用します。アプリケーションのページ・コンテンツをモジュール化するためのメカニズムです。
<code>&lt;csp:loop&gt;</code>	<ul style="list-style-type: none"> <li>• <code>counter</code>: 各繰り返しに伴って変更する値を保持する変数。</li> <li>• <code>from</code>: <code>counter</code> の初期値。</li> <li>• <code>to</code>: <code>counter</code> の最終値。 <code>counter</code> がこの値に達すると、ループの繰り返しが停止します。</li> <li>• <code>step</code>: 各繰り返しにおける <code>counter</code> のインクリメント。</li> </ul>	ObjectScript または Basic の <code>for</code> ループの機能を提供します。コード・ブロックを一定の回数だけ繰り返します。
<code>&lt;csp:while&gt;</code>	<ul style="list-style-type: none"> <li>• <code>condition</code>: ループの制御条件。条件の評価が偽を返すと、ループは終了します。</li> <li>• <code>counter</code>: 各ループの繰り返しに伴って値が変更するオプションの変数。開始値は 1 で、インクリメントは 1 です。</li> </ul>	ObjectScript または Basic の <code>while</code> ループの機能を提供します。制御条件の評価が偽を出力するまで、コード・ブロックを繰り返します。 <code>condition</code> の評価は、 <code>counter</code> がインクリメントする前に行われることに注意してください。 <code>counter</code> の変数は、ループの実行回数をカウントします。

## その他の CSP タグ

タグ	属性	説明
<csp:if>、 <csp:elseif>、 <csp:else>	<ul style="list-style-type: none"> <li>• <b>condition</b>:コード・ブロックの制御条件。 <b>condition</b> の評価が真を出力すると、コード・ブロックが実行されます。</li> </ul>	ObjectScript または Basic の <b>if</b> 、 <b>elseif</b> 、 <b>else</b> の条件構造の機能を提供します。<csp:if>の範囲内につかの<csp:elseif>と最後の<csp:else>を入れ子にします。
<csp:query>	<ul style="list-style-type: none"> <li>• <b>name</b>:返された <a href="#">%ResultSet</a> オブジェクトの参照。</li> <li>• <b>classname</b>:クエリを含むクラスの名前。</li> <li>• <b>queryname</b>:クエリの名前。</li> <li>• <b>P1,P2,...,PN</b>:クエリ・パラメータ値。</li> </ul>	Cachéクラスの定義済みクラス・クエリを実行します。クエリの結果を含む <a href="#">%ResultSet</a> オブジェクトをCSPコードに返します。 <csp:while>を使用して、結果に繰り返し処理を行います。
<csp:comment>	N/A	CSP ページ内にコメントを指定します。コメントは、クライアントに送信されるコンテンツに含まれません。クライアントに送信されるコメントを含めるには、HTML スタイルのコメント区切り文字<!-- --> を使用します。

### Note:

CSPタグの詳細は、Cachéドキュメント: [\[Caché 開発ガイド\]-\[Caché Server Page\(CSP\)の使用法\]-\[CSPにおけるタグを使用した開発\]](#) を参照してください。CSPタグのリストについては、Cachéドキュメント: [\[Caché 開発リファレンス\] - \[CSP HTMLタグ・リファレンス\]-\[CSP タグ\]](#) を参照してください。

## CSP タグ: &lt;csp:include&gt;

<csp:include>を使用して、ある CSP ページのコンテンツを別のページに含めるには以下の手順を実行します。

1. Caché スタジオを使用して、Menu.CSP という名前の新しい CSP ページを作成します。このページの<body>と</body>との間に以下のコンテンツを追加します。

```
<body>
<h3>
<a href="http://www.intersystems.com/cache">Caché</a>
</h3>
<h3>
<a href="http://www.intersystems.com/ensemble/index.html">Ensemble</a>
</h3>
<h3>
<a href=
"http://www.intersystems.com/cache/devcorner">Developer's Corner</a>
</h3>
<h3>
<a href=
"http://www.intersystems.com/cache/education/elearning/index.html">
E-Learning</a></h3>
<hr style="color:red;"></hr>
</body>
```

2. Caché スタジオで MySamplePage.CSP を開きます。<body> と</body>との間に以下の<csp:include>タグを追加します。page 属性の値は、挿入先の CSP ページの URL です。<body> と</body>との間にあるその他のコードはすべて削除できます。

```
<body>
<csp:include page="Menu.CSP"/>
</body>
```

3. **[表示]**→**[ブラウザで表示]**をクリックすると、Web ブラウザに MySamplePage.CSP が表示されます。CSP サーバは Menu.CSP を自動的にコンパイルし、そのコンテンツを MySamplePage.CSP に追加します。

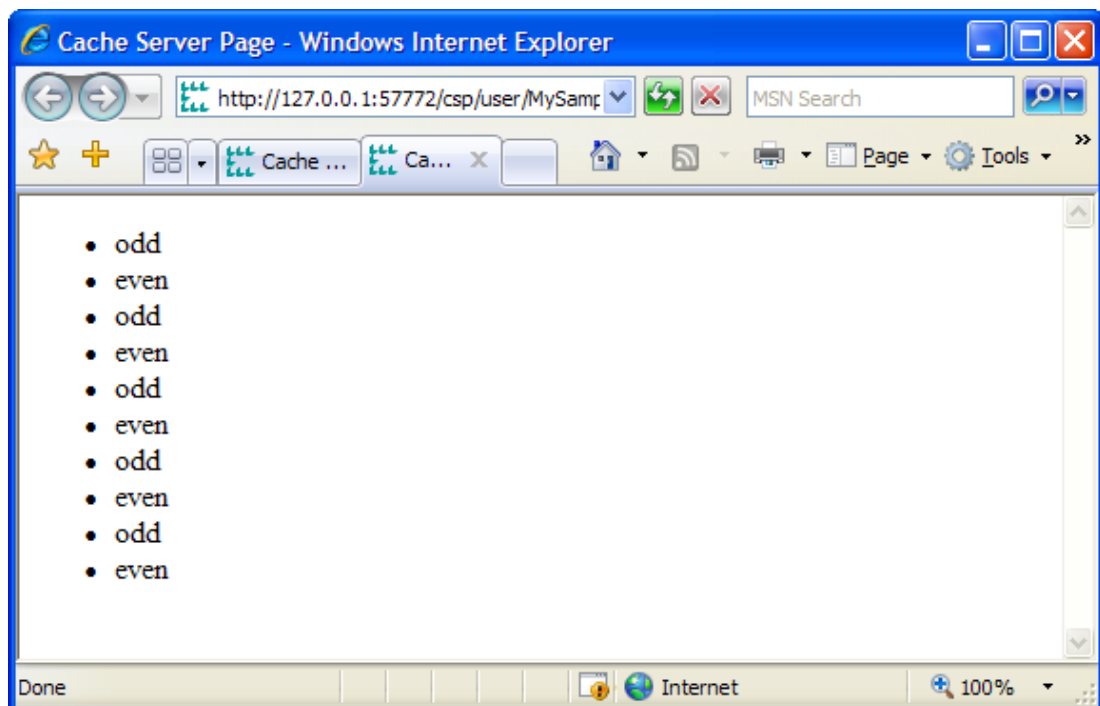
## CSP タグ: &lt;csp:loop&gt;

<csp:loop>タグは、指定した回数だけコード・ブロックを繰り返します。

MySamplePage.CSP の<body>と</body>との間に以下のコードを追加します。このコードは、**OddEvenTest** を 10 回実行して、その結果を箇条書きリストで表示します。 <body>と</body>との間にあるその他のコードはすべて削除できます。

```
<body>
<ul>
<csp:loop counter="i" from="1" to="10" step="1">
<li>#(..OddEvenTest(i))#</li>
</csp:loop>
</ul>
<script language="cache" method="OddEvenTest" arguments="val1:%Integer"
returntype="%String">
  If (val1#2=0) { quit "even" }
  Else {quit "odd"}
</script>
</body>
```

**[表示]**→**[ブラウザで表示]**をクリックすると、Webブラウザに MySamplePage.CSP が表示されます。

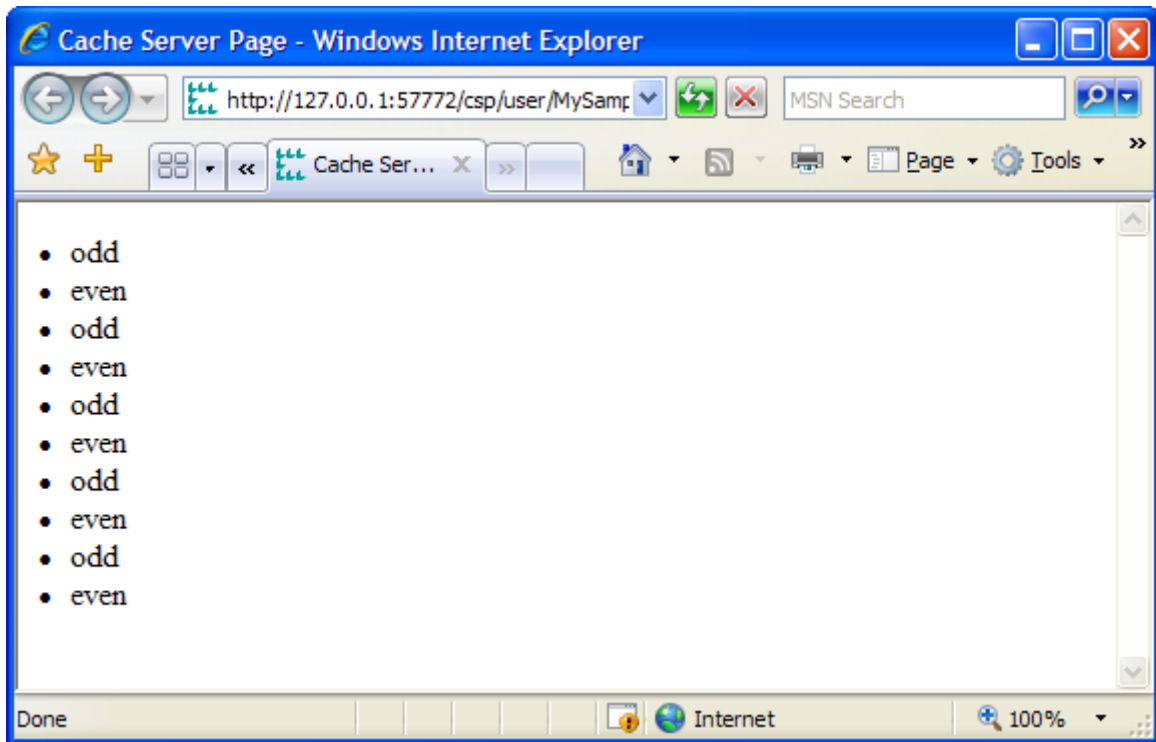


## CSP タグ: &lt;csp:while&gt;

<csp:while>タグは、その制御条件の評価が真を返す間、コード・ブロックを繰り返します。MySamplePage.CSP の<body>と</body>との間に以下のコードを追加します。このループは、数字 1~10 を OddEvenTest に送信しながら OddEvenTest を 10 回実行します。<body>と</body>との間にあるその他のコードはすべて削除できます。

```
<body>
<script language="Cache" runat="Server">
Set i=0
</script>
<csp:while condition="i<10" counter="j">
<li>#(. OddEvenTest(j))#</li>
<script language="Cache" runat="Server">
Set i=i+1
</script>
</csp:while>
</ul>
<script language="cache" method="OddEvenTest" arguments="val1:%Integer"
returntype="%String">
  If (val1#2=0) { quit "even" }
  Else {quit "odd"}
</script>
</body>
```

**[表示]**→**[ブラウザで表示]**をクリックすると、Webブラウザに MySamplePage.CSP が表示されます。



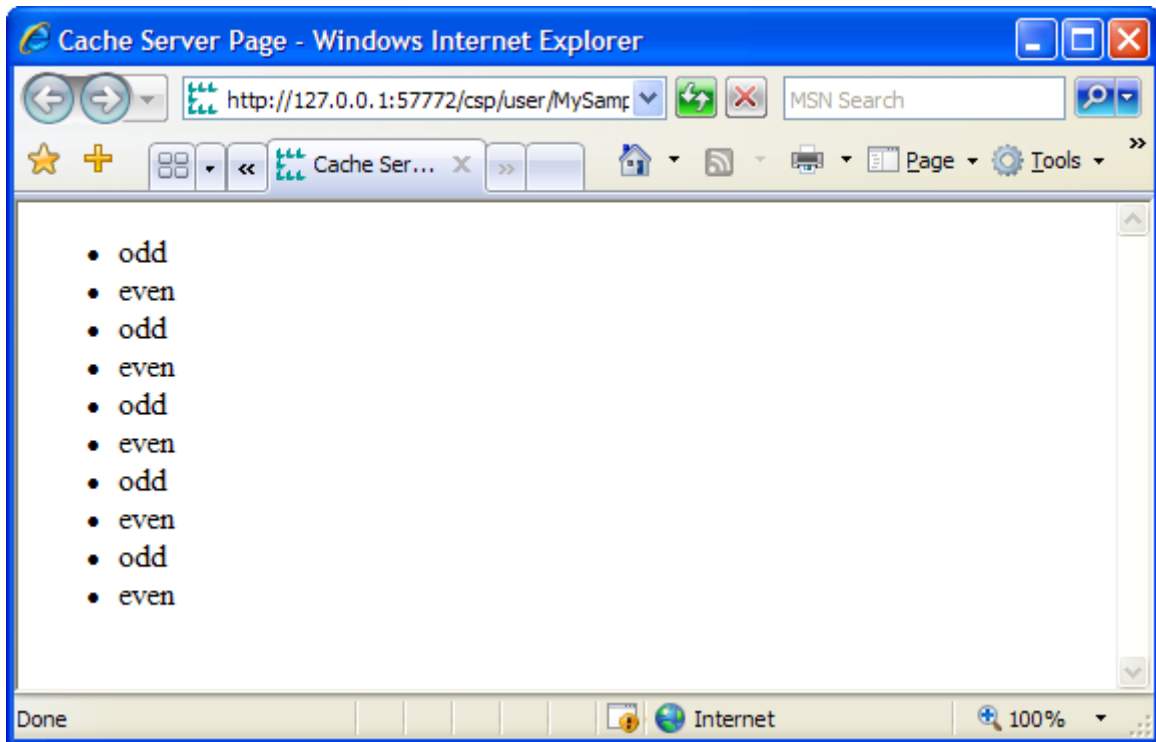
## CSP タグ: &lt;csp:if&gt;

<csp:if>、<csp:elseif>、<csp:else>タグは、CSP ページのフロー内に条件構造を作成します。

MySamplePage.CSP の<body>と</body>との間に以下のコードを追加します。このコードは、条件文を使用して **OddEvenTest** の機能を再作成します。条件文は 10 回実行するループ内にあります。<body>と</body>との間にあるその他のコードはすべて削除できます。

```
<body>
<ul>
<csp:loop counter="i" from="1" to="10" step="1">
  <csp:if condition="i#2=0">
    <li>even</li>
  <csp:else condition="i#2' =0">
    <li>odd</li>
  </csp:else>
</csp:if>
</csp:loop>
</ul>
</body>
```

**[表示]**→**[ブラウザで表示]**をクリックすると、Webブラウザに MySamplePage.CSP が表示されます。



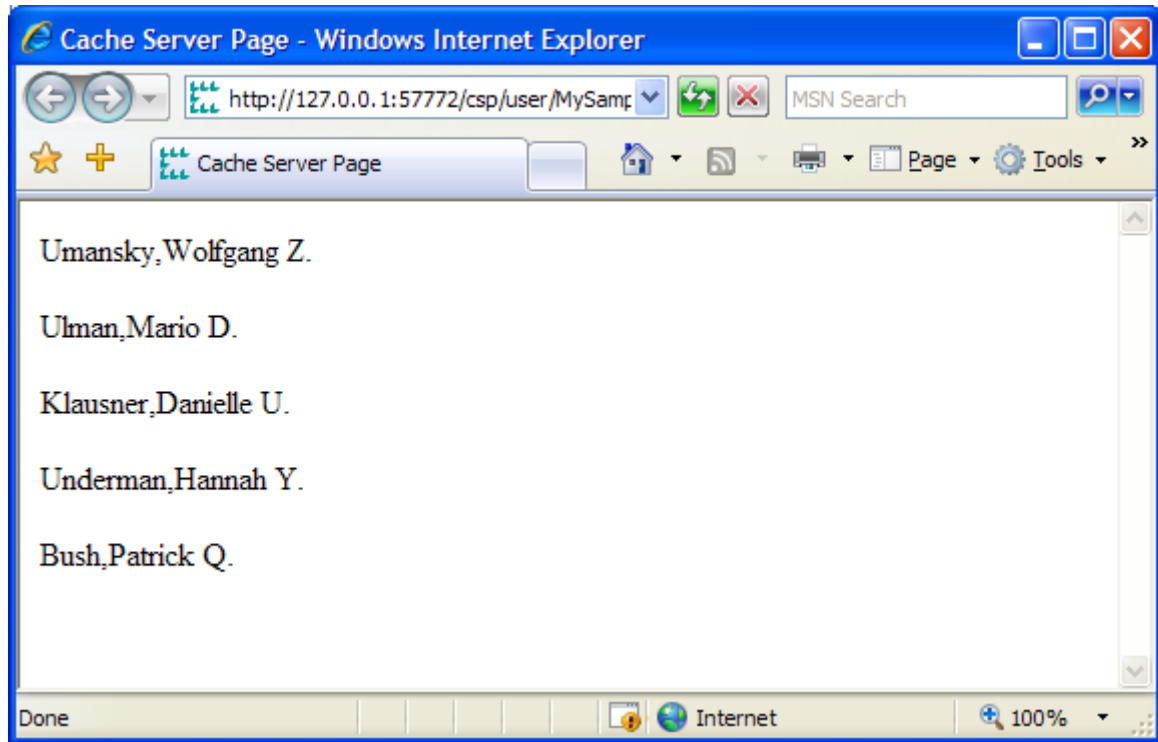
## CSP タグ: &lt;csp:query&gt;

<csp:query>タグを使用して、CSPクラスで定義するクラス・クエリを実行します。<csp:while>を使用して、クエリで返される`%ResultSet`オブジェクトに繰り返し処理を行います。

MySamplePage.CSPの<body>と</body>との間に以下のコードを追加します。このコードは<csp:query>を使用して、`CSPTutorial.Contact`で定義する `ByContactType`クラス・クエリを実行し、<csp:while>を使用して、クエリで返される`%ResultSet`オブジェクトに繰り返し処理を行います。 <body>と</body> との間にあるその他のコードはすべて削除できます。

```
<body>
<csp:query name="query" classname="CSPTutorial.Contact"
queryname="ByContactType" P1="Business">
<csp:while condition="query.Next()">
<p>#(query.Data("Name"))#</p>
</csp:while>
</body>
```

**[表示]**→**[ブラウザで表示]**をクリックすると、Web ブラウザに MySamplePage.CSP が表示されます。



**Note:**

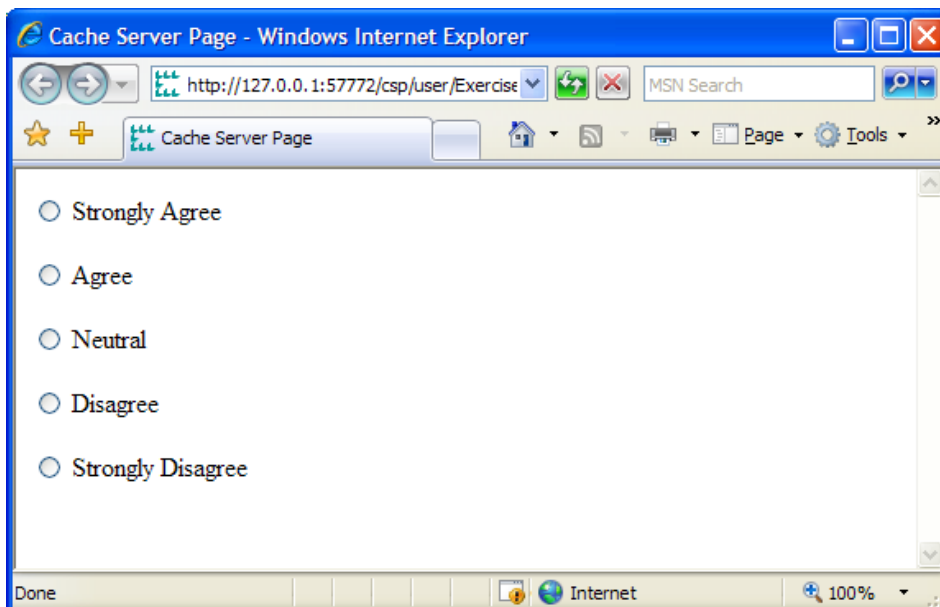
クラス・クエリの詳細は、Cacheドキュメント: [\[Cache 開発ガイド\]](#) - [\[Cacheオブジェクトの使用法\]](#) - [\[クラス・クエリ\]](#)を参照してください。

## 演習

演習 1: [Strongly Agree]、[Agree]、[Neutral]、[Disagree]、[Strongly Disagree] というラベルの一連のラジオ・ボタンを出力する CSP ページを作成します。この CSP ページでは以下の要素を使用する必要があります。

- 5 回繰り返す `<csp:loop>` タグ。繰り返しごとに、`<input type="Radio">` タグを出力します。
- メソッドに渡された整数の値に応じて、“Strongly Agree”、“Agree”、“Neutral”、“Disagree”または“Strongly Disagree”を返す **AddValue** という名前の関数。
- 2 つの CSP 実行時式。これらの式には、Value 属性の値 (“Strongly Agree”、“Agree” など)と、各 `<input type="Radio">` タグのコンテンツを入力します。式は、**AddValue** を呼び出し、このメソッドに `<csp:loop>` カウンタの現在の値を渡します。

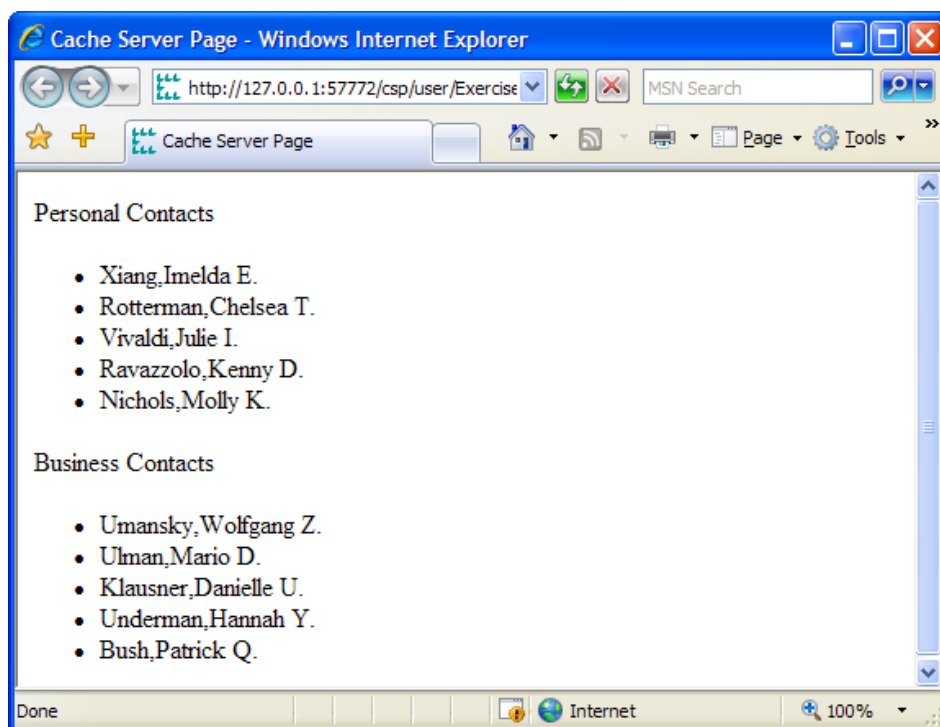
このページの出力は以下のようになります。



演習 2: すべての連絡先の名前をデータベースに出力する CSP ページを作成します。連絡先は、ページ上で Personal と Business の 2 つのグループに分類されます。この CSP ページでは以下の要素を使用する必要があります。

- 1 つの `<csp:query>` タグ。このタグは、**Contact** クラスの **ByContactType** クエリを使用します。
- クエリの結果セットに繰り返し処理を行い、連絡先の名前を表示する `<csp:while>` タグ。
- 連絡先のタイプごとに `<csp:query>` を 1 回実行する `<csp:loop>` タグ。
- 整数の引数の値に応じて、“Business”または“Personal”を返す関数。
- 必要に応じた CSP 実行時式。

このページの出力は以下のようになります。



**Note:**

演習 2 を実行するには、USER ネームスペースに **CSPTutorial.Contact** をインストールして生成する必要があります。PartIStarter.xml にはクラス定義があります。インストールおよび生成の手順については、[Appendix A: チュートリアル・ファイルのインストール] を参照してください。

PartIExerciseSoln.xml には、演習 1 および演習 2 の解答があります。このファイルの場所と解答のインストールの手順については、[Appendix A: チュートリアル・ファイルのインストール] を参照してください。

## 要約

このチュートリアル第 1 章では、以下について学習しました。

- CSP プラットフォームの各種コンポーネントの役割。
- CSP プラットフォームのコンポーネントの構成。
- CSP ページのライフサイクル。
- CSP ページから Caché クラスへの変換のリレーションシップ。
- Caché スタジオを使用した CSP ページの作成。
- 各種 CSP 要素(式、スクリプトおよび CSP マークアップ・タグなど) の使用。

## はじめに

このチュートリアル第 2 章では、オブジェクト・バインディング、フォーム・ウィザード、検索ページ、ページ指示文の使用、状態の管理、ハイパーイベント、ページのリダイレクトといった CSP プログラミングの基本的なトピックについて説明します。この章には、演習や例を収めたファイルがいくつか用意されています。この章の学習を始める前に、これらのファイルをインストールする必要があります。手順については、下記のメモを参照してください。

この章の学習を終えると、以下のことを実行できるようになります。

- ページ指示文を使用してページ言語を設定する。
- オブジェクトをフォームや CSP ページに結合する。
- Caché スタジオの Web フォーム・ウィザードを使用してオブジェクトに結合する Web フォームを作成する。
- Caché クラスのインスタンスを検索するカスタム検索ページを作成する。
- CSP ページにクエリを追加する。
- CSP アプリケーションのユーザの状態を管理する。
- ハイパーイベントを使用して、CSP ページからサーバ側のメソッドを呼び出す。

### Note:

この章に付随するファイルのインストール手順については、[Appendix A: チュートリアル・ファイルのインストール] を参照してください。

## ページ指示文

ページ指示文を使用して、CSP ページのページ言語を指定します。 Caché ObjectScript が既定のページ言語です。 ページ指示文は、CSP ファイルの最上部に表示されます。 以下の指示文では、ページ言語を Caché Basic に設定しています。

```
<%@Page Language="Basic" %>
```

ページ言語は、以下のために使用する言語を決定します。

1. 生成されたコード。
2. CSP タグ。
3. 実行時式 — #(**expr**)#式。
4. スクリプト — ページ言語と一致する `language` 属性の値。

ページ言語が Caché ObjectScript の場合：

1. メソッドは、Caché ObjectScript または Caché Basic のいずれかを使用できます。  
`script` タグの `attribute` が言語を識別します。
2. コンパイル時の式 `##(<expr>)##` は Caché ObjectScript を使用します。
3. CSP ページに、埋め込み SQL、埋め込み HTML、埋め込み JavaScript を使用できません。

ページ言語が Caché Basic の場合：

1. メソッドは、Caché ObjectScript または Caché Basic のいずれかを使用できます。  
`script` タグの `attribute` が言語を識別します。
2. コンパイル時の式 `##(<expr>)##` でも Caché ObjectScript を使用する必要がありません。
3. CSP ページに、埋め込み SQL 埋め込み HTML 埋め込み JavaScript は使用できません。

### Note:

このチュートリアル例では、既定のページ言語である Caché ObjectScript を使用します。 Caché ObjectScript についてさらに学習するには、Caché ドキュメント：[\[Caché 開発ガイド\] – \[Caché ObjectScript の使用法\]](#) および Caché ドキュメント：[\[Caché チュートリアル\] – \[Caché ObjectScript チュートリアル\]](#) を参照してください。 Caché Basic についてさらに学習するには、Caché ドキュメント：[\[Caché 開発ガイド\] – \[Caché Basic の使用法\]](#) および Caché ドキュメント：[\[Caché チュートリアル\] – \[Caché Basic チュートリアル\]](#) を参照してください。

## オブジェクト・バインディング

CSP オブジェクト・バインディングを使用すると、CSP ページではページ全体で使用する永続オブジェクトを開くことができます。オブジェクトをページに結合すると、そのプロパティとメソッドをページ全体で使用できます。<csp:object>タグを使用して、オブジェクトをページに結合します。構文は以下のようになります。

```
<csp:object name="oref" classname="class" objid="id">
```

このタグは以下の属性をサポートします。

- name — ページ全体でオブジェクトの参照に使用する変数。
- classname — オブジェクトのクラスの名前。
- objid — オブジェクトの ID。この属性の値の入力はオプションです。値を入力すると、ページはその値を引数として使用し、クラスの%OpenIdメソッドを実行します。値を入力しない場合、クラスの%Newメソッドを実行します。

## オブジェクト・バインディングの例

この例では、以下を実行する簡単な CSP ページを作成します。

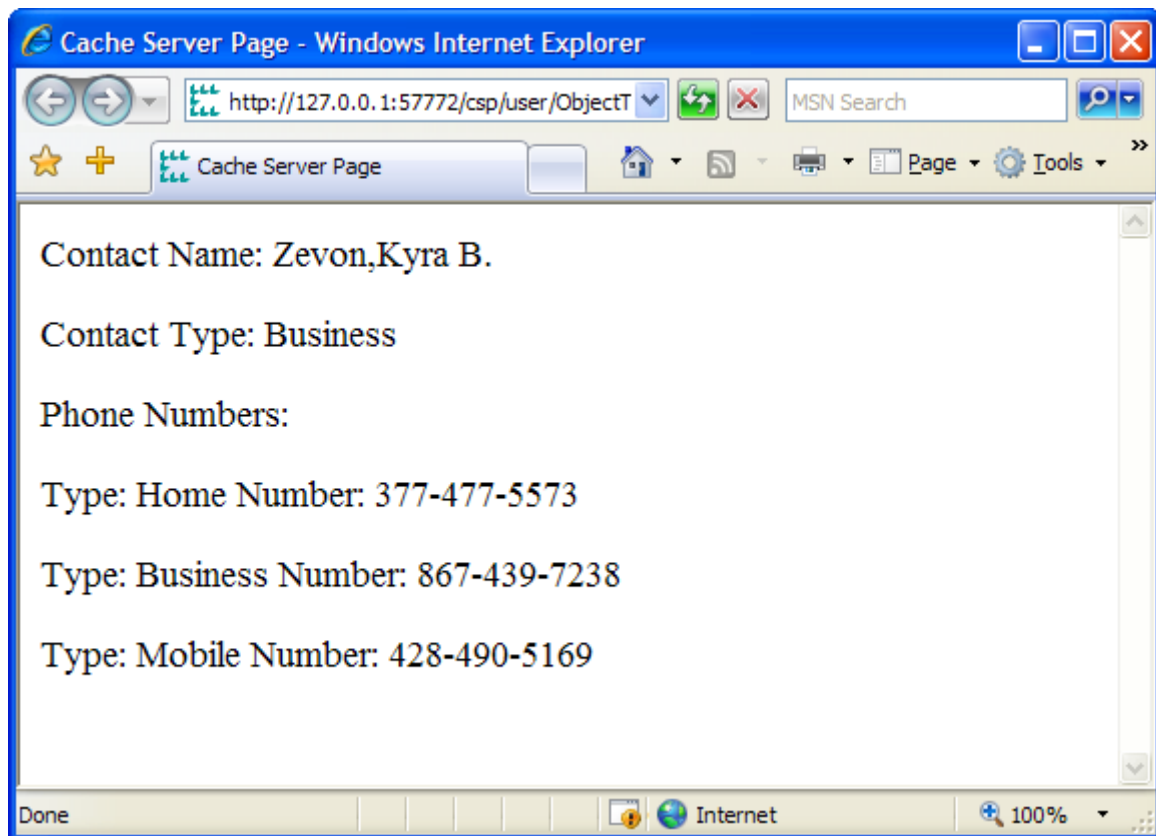
- `<csp:object>`を使用して、ID の値が 1 の **Contact** オブジェクトを開きます。
- オブジェクトのプロパティの値を表示します。

手順は以下のようになります。

1. Caché スタジオの CSP ウィザードを使用して、csp/user アプリケーションに新しい CSP ページを作成します。 ページの名前を ObjectTest.csp にします。
2. `<body>`タグとそのコンテンツを以下のように変更します。 コードが CSP 式を使用してオブジェクトのプロパティの値にアクセスする点に注意してください。

```
...
<body style="font-size:120%">
<csp:object name="contact" classname="CSPTutorial.Contact" objid="1"/>
<p>Contact Name: #(contact.Name)#</p>
<p>Contact Type: #(contact.ContactType)#</p>
<p>Phone Numbers:</p>
<script language="Cache" runat="Server">
for i=1:1:contact.PhoneNumbers.Count()
{
  Set pn=contact.PhoneNumbers.GetAt(i)
  Write "<p>Type: "_pn.PhoneNumberType_" Number: "_pn.Number_"</p>"
}
</script>
</body>
...
```

3. **[表示]**→**[ブラウザで表示]**をクリックすると、ページが表示されます。



**Note:**

標準の Caché インストールの場合、この CSP アプリケーションを構成する必要はありません。Caché スタジオの CSP ウィザードを使用した CSP ページの作成の演習ガイドについては、このチュートリアル第 1 章 - Chapter 10: の「CSP サンプル・ページ」セクションを参照してください。CSP 式の詳細は、このチュートリアル第 1 章 - Chapter 11: の「CSP 要素: 式」を参照してください。

## フォーム・バインディング

CSPは、HTMLフォームや他の入力コントロールの `cspbind` 属性をサポートします。この属性を使用して、オブジェクトやそのプロパティを HTML コントロールに結合します。これにより、コントロールでは、結合したオブジェクトのプロパティの値を表示および更新できます。ページがロードされると、コントロールはプロパティの値を表示します。フォームが送信されると、プロパティの値が更新されます。

オブジェクトを HTML フォームに、オブジェクトのプロパティをフォームの入力コントロールに結合する手順は以下のとおりです。

1. `<csp:object>`を使用して、ページでオブジェクトを開きます。
2. `cspbind` 属性を`<form>`タグに追加します。 `cspbind` の値は、`<csp:object>`タグの `name` 属性の値と一致します。
3. `cspbind` 属性を`<input>`タグに追加します。 `cspbind` 属性の値は、結合したプロパティの名前と一致します。

次に例を示します。

```
<csp:object name="contact" classname="CSPTutorial.Contact" objid="1"/>
<form name="MyForm" cspbind="contact">
<input type="text" name="txtContactType" cspbind="ContactType">
```

オブジェクトをフォームに結合すると、CSP コンパイラは以下のコードを自動的に生成します。

- 入力必須フィールドの確認やその他の基本的なフォームの検証を実行する、クライアント側の JavaScript。
- 結合されたオブジェクトを保存するためにサーバ側メソッドを呼び出す、クライアント側の JavaScript。
- オブジェクト ID 値を含む、フォームの非表示フィールド OBJID。
- フォームに入力されたデータを検証して保存する、サーバ側のメソッド。

### Note:

`cspbind`をサポートするHTML入力要素の全リストについては、Cachédキュメント: [\[Caché 開発ガイド\]](#) - [\[Caché Server Pages \(CSP\)の使用法\]](#) - [\[データベース・アプリケーションの構築\]](#) - [\[フォームへのデータの結合\]](#)の説明を参照してください。

## フォーム・ウィザードの概要

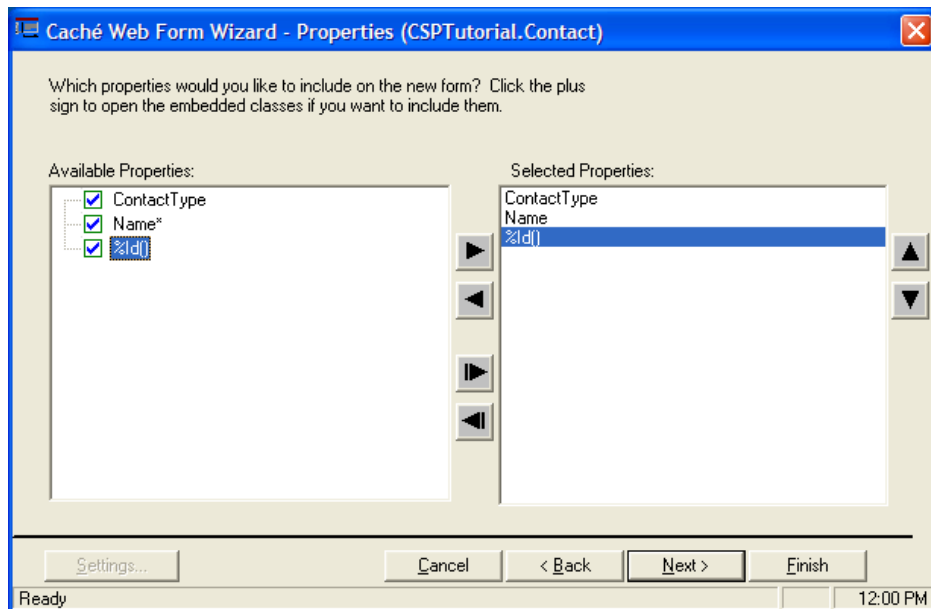
Caché スタジオのフォーム・ウィザードは、オブジェクトとそのプロパティの HTML フォームへの結合に必要なコーディング作業を自動化します。フォーム・ウィザードは、結合に必要なコードの他に以下も生成します。

- CSP<csp:search>タグ。これらのタグは、データベースの検索機能を実装します。ウィザードは、この検索機能を有効にする **[検索]** ボタンもフォームに配置します。
- 結合されたクラスの新しいインスタンスを作成する JavaScript 関数 **form\_new** と各種ヘルパー関数。この関数は、ウィザードによってフォームに配置される **[クリア]** ボタンで呼び出され、フォームの入力コントロールのコンテンツを消去します。
- 結合されたクラスの開いているオブジェクトを更新および保存する JavaScript 関数 **form\_save** と各種ヘルパー関数。ウィザードはこのメソッドを呼び出す **[保存]** ボタンもフォームに配置します。

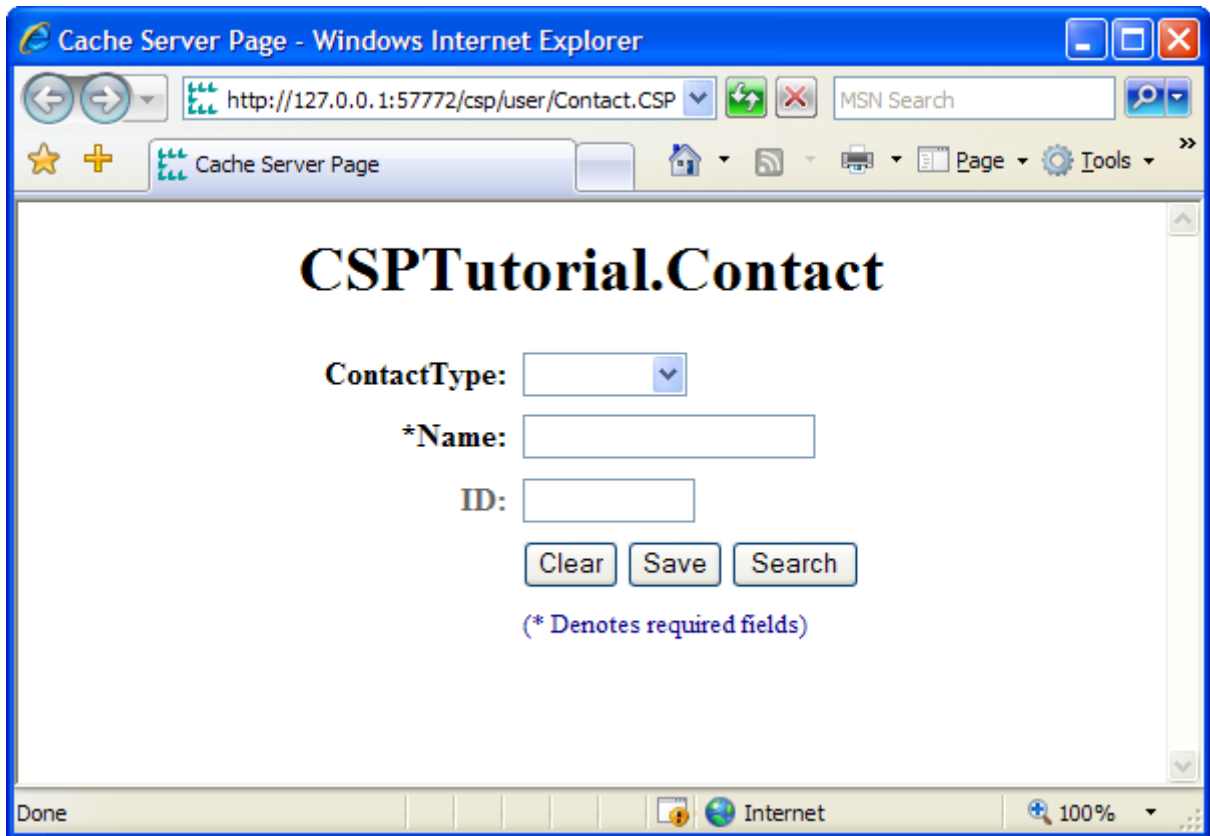
## フォーム・ウィザードの例

フォーム・ウィザードを使用して **CSPTutorial.Contact** クラスのフォームを作成するには、以下の手順を実行します(クラスの場所については下記のメモを参照してください)。

1. Caché スタジオを開き、USER ネームスペースに接続します。Contact.CSP という名前の新しい CSP ページを作成します。
2. ページの<body></body>タグの間にスタジオのカーソルを置きます。
3. **[挿入]**→**[フォーム・ウィザード]**をクリックして、フォーム・ウィザードを起動します。
4. **[よろこ]**画面で**[次へ]**をクリックします。
5. **[クラス]**画面で **CSPTutorial.Contact** をクリックします。 **[次へ]**をクリックします。
6. **[プロパティ]**画面で、フォームに表示させるすべてのプロパティをクリックします。この場合、**[ContactType]**、**[Name]**、および**[%Id()**をすべてクリックします。



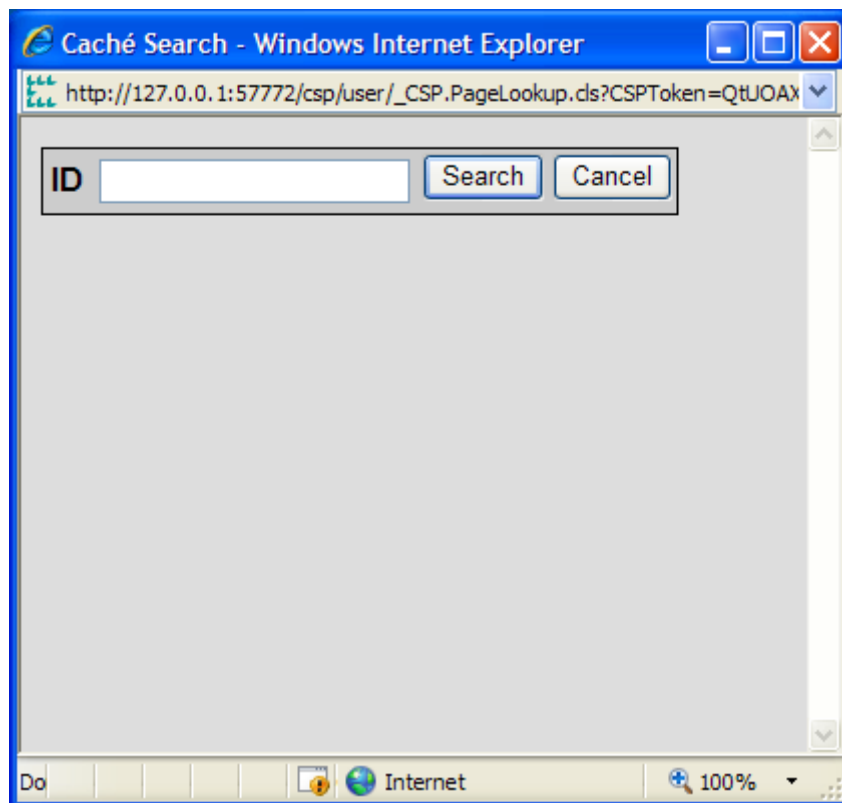
7. **[完了]**をクリックします。ウィザードは、Web フォームのすべてのコードを CSP ページに追加します。
8. **[表示]**→**[ブラウザで表示]**をクリックして、以下のフォームをテストします。



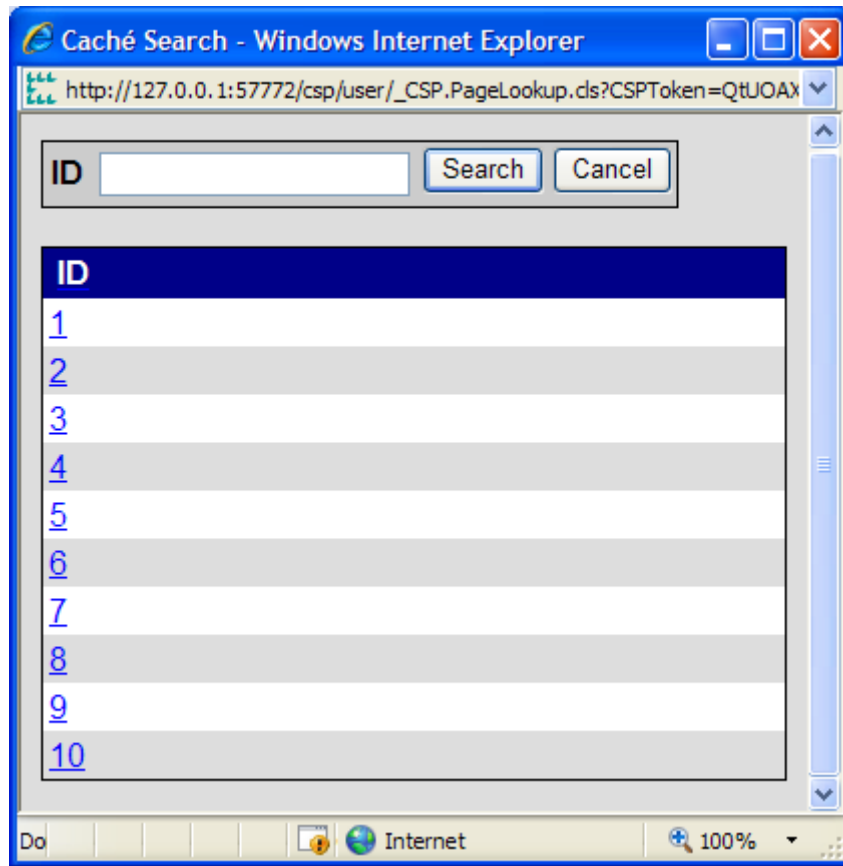
## 検索ページ

フォーム・ウィザードで生成したフォーム上の**[検索]**ボタンをクリックすると、検索ページが表示されます。この検索ページは、フォーム・ウィザードが CSP ページに挿入する<csp:search>によって生成されます。これらのタグによって、CSP コンパイラは、検索ページを表示するクライアント側の JavaScript 関数を生成します。検索ページを使用すると、フォームに関連付けられた永続クラスのインスタンスを検索できます。既定の検索ページでは、ID プロパティのみによる検索が可能です。検索ページには、検索結果がハイパーリンクのテーブルとして表示されます。リンクをクリックすると、対応するオブジェクトがフォームにロードされます。

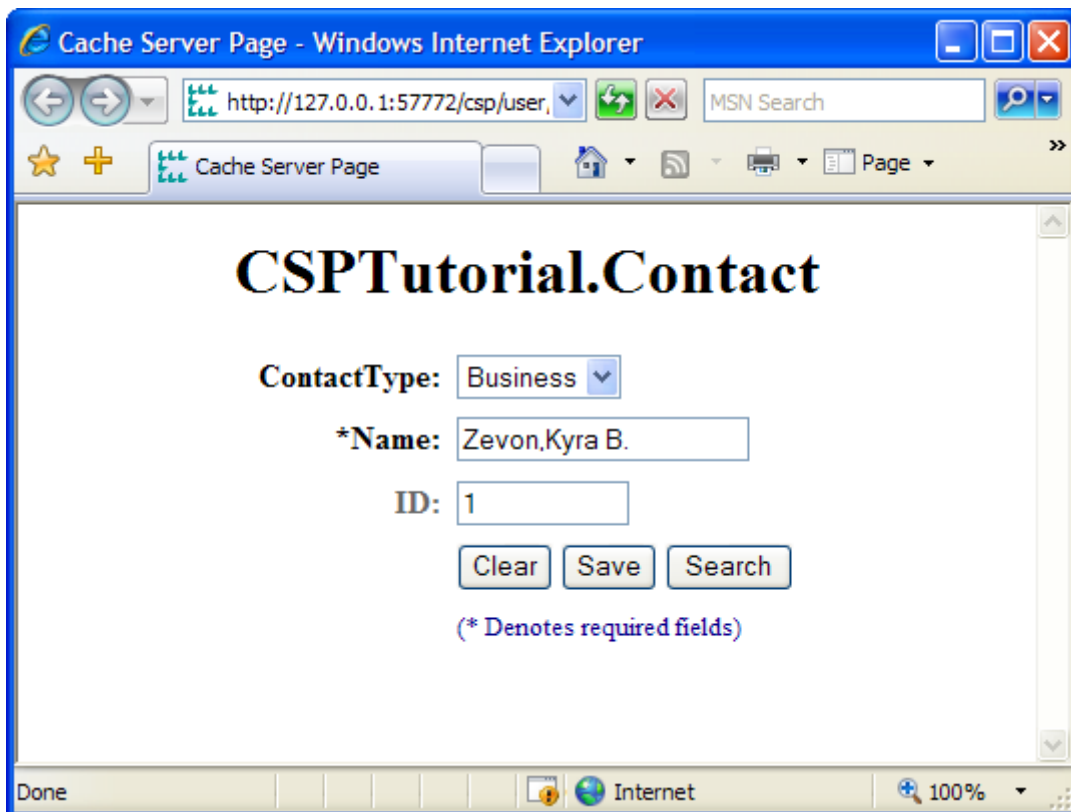
Contact.CSP の既定の検索ページは以下のとおりです。



**[検索]**をクリックすると、データベースの **Contact** ID の値がすべてリスト表示されます。



1 をクリックすると、**Contact.CSP** が開き、フォームにロードされた ID 値が 1 の **Contact** のプロパティ値が表示されます。



## 検索ページのカスタマイズ

<csp:search>の属性を使用して、既定の検索ページをカスタマイズします。以下で、いくつかの属性について説明します。

### 検索タグ属性

属性	説明
NAME	必須項目。属性値は、検索ページを呼び出す、生成されたクライアント側の JavaScript 関数の名前です。
ONSELECT	ポップアップ検索ページの場合、属性値は、ユーザが検索結果を選択する場合に呼び出されるクライアント側の JavaScript 関数の名前です。
OPTIONS	属性値は、コンマで区切られた、検索ページのオプションのリストです。以下のようなオプションがあります。 <ul style="list-style-type: none"> <li>• ポップアップ — 検索ページは、ポップアップ・ウィンドウで表示されます。</li> <li>• 述語 — 検索ページに、検索述語のドロップダウン・リストが表示されます。</li> <li>• 述語以外 — 検索ページに、検索述語のドロップダウン・リストは表示されません。</li> </ul>
ORDER	属性値は、検索結果を並べ替える SELECT フィールドの名前です。
SELECT	属性値は、検索結果に表示するフィールドのコンマ区切りのリストです。値を指定しない場合、WHERE 属性の値に指定したフィールドが使用されます。
WHERE	必須項目。属性値は、コンマで区切られた、検索する検索ページのフィールドのリストです。

#### Note:

<csp:search>タグの属性の全リストについては、Cachédキュメント: [\[Caché 開発ガイド\]](#) - [\[Caché Server Pages \(CSP\)の使用方法\]](#) - [\[データベース・アプリケーションの構築\]](#) - [\[<csp:search> タグを持つ CSP 検索ページ\]](#) の説明を参照してください。

## カスタム検索ページの例

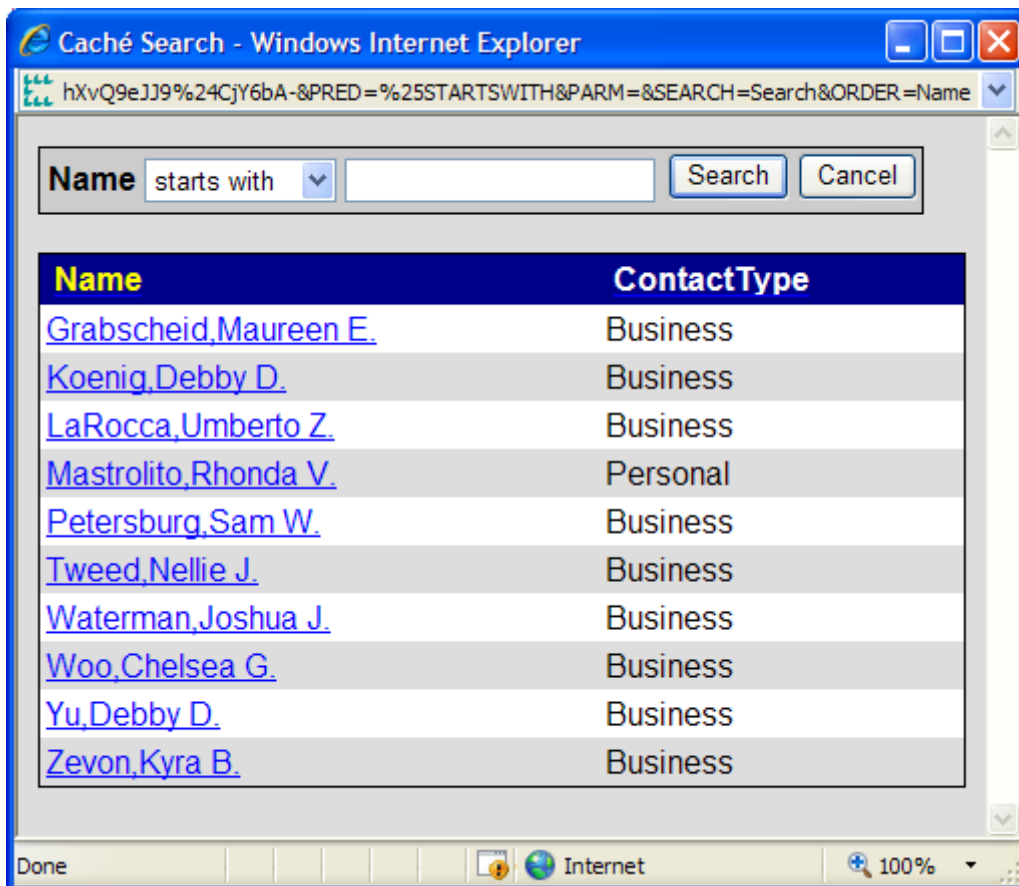
Contact.csp に追加する<csp:search>タグの属性を以下のように変更します。

```
<csp:search name="form_search" classname="CSPTutorial.Contact"
  Where="Name" select="Name,ContactType" options="popup,predicates"
  onselect="update" order="Name">
```

この変更により、既定の検索ページは以下のように変わります。

- 検索ページは、**Contact** Name フィールドで検索します。
- 検索結果には、各検索結果の Name と ContactType フィールドが表示されます。
- 検索ページには、検索述語のドロップダウン・リストが表示されます。
- 検索結果は、Name フィールドを基準に並べ替えられます。

変更された検索ページは以下のようになります。



## クエリ

CSP は、Cache にクエリを実行するいくつかの方法をサポートしています。

## Cache にクエリを実行する方法

クエリ・タイプ	CSP タグと属性	説明
クラス・クエリ	<code>&lt;csp:query&gt;</code> <ul style="list-style-type: none"> <li>• <code>name</code> — 返された <a href="#">%ResultSet</a> オブジェクトの参照</li> <li>• <code>classname</code> — クエリを含むクラスの名前</li> <li>• <code>queryname</code> — クエリの名前</li> <li>• <code>P1,P2,...,PN</code> — クエリ・パラメータ値</li> </ul>	Cache クラスで定義するクラス・クエリを実行します。 <a href="#">%ResultSet</a> オブジェクトを返します。 <code>&lt;csp:while&gt;</code> を使用して、結果セットに繰り返し処理を行います。
ダイナミック SQL	<code>&lt;script&gt;</code> <ul style="list-style-type: none"> <li>• <code>language</code> — 値は <code>sql</code></li> <li>• <code>name</code> — 返された <a href="#">%ResultSet</a> オブジェクトの参照</li> <li>• <code>P1,P2,...,PN</code> — クエリ・パラメータ値</li> <li>• <code>mode</code> — クエリの実行時モード (Logical、ODBC、Display または System)。</li> </ul>	CSP コードのインライン・クエリを実行します。 <a href="#">%ResultSet</a> オブジェクトを返します。 <code>&lt;csp:while&gt;</code> を使用して、結果セットに繰り返し処理を行います。

## Caché にクエリを実行するその他の方法

クエリ・タイプ	CSP タグと属性	説明
Embedded SQL (埋め込み SQL)	<p><code>&lt;script&gt;</code></p> <ul style="list-style-type: none"><li><code>language</code> — カーソル・ベースの場合、値は <code>sql</code>。非カーソル・ベースの場合、値は <code>esql</code>。</li><li><code>cursor</code> — カーソル・ベースの場合のみ、カーソルの参照。</li><li><code>mode</code> — クエリの実行時モード (<code>Logical</code>、<code>ODBC</code>、<code>Display</code> または <code>System</code>)。</li></ul>	<p>カーソル・ベースおよび非カーソル・ベースの両方をサポートします。非カーソル・ベースの場合、<code>language</code> 属性のみがサポートされます。カーソル・ベースの場合、<code>&lt;csp:while&gt;</code> を使用して結果セットに繰り返し処理を行います。</p> <p><code>&lt;csp:while&gt;</code> タグは、<code>cursor</code> および <code>into</code> 属性をサポートします。値は、<code>&lt;script&gt;</code> タグの値と同じです。<code>into</code> の値は、クエリ結果を含む変数のリストです。</p>

### Note:

`<csp:query>` タグを使用してクラス・クエリを実行する例については、このチュートリアル第 1 章 / Chapter 19: の [CSP タグ: `<csp:query>`] セクションを参照してください。

CSPでのSQLの使用についてさらに学習するには、Cachéドキュメント: [\[Caché 開発ガイド\]](#) - [\[Caché Server Pages \(CSP\)の使用法\]](#) - [\[CSPにおけるタグを使用した開発\]](#) - [\[SQL <script> タグ\]](#)に関する説明を参照してください。

Cachéでの埋め込みSQLについてさらに学習するには、Cachéドキュメント: [\[Caché 開発ガイド\]](#) - [\[Caché SQLの使用法\]](#) - [\[埋め込みSQL\]](#) を参照してください。 CachéでのダイナミックSQLについてさらに学習するには、Cachéドキュメント: [\[Caché 開発ガイド\]](#) - [\[Caché SQLの使用法\]](#) - [\[ダイナミックSQL\]](#)を参照してください。

## ダイナミック・クエリの例

CSP でダイナミック・クエリを使用するには、以下の手順を実行します。

1. Caché スタジオを使用して、csp/user に MySamplePage2.CSP という名前の新しい CSP ページを作成します。
2. ページに以下のスクリプトを追加します。このスクリプトは、Contact 値が入力した値と一致する各 **PhoneNumber** の PhoneNumberType および Number の値を取得するクエリを定義します。スクリプトは%request から Contact 値を取得します。この方法についてはチュートリアルの後半で説明します。

```
<script language="sql" name="rs" p1="#($Get(%request.Data("ID",1)))#" mode="display">  
SELECT PhoneNumberType, Number FROM CSPTutorial.PhoneNumber WHERE Contact = ?  
</script>
```

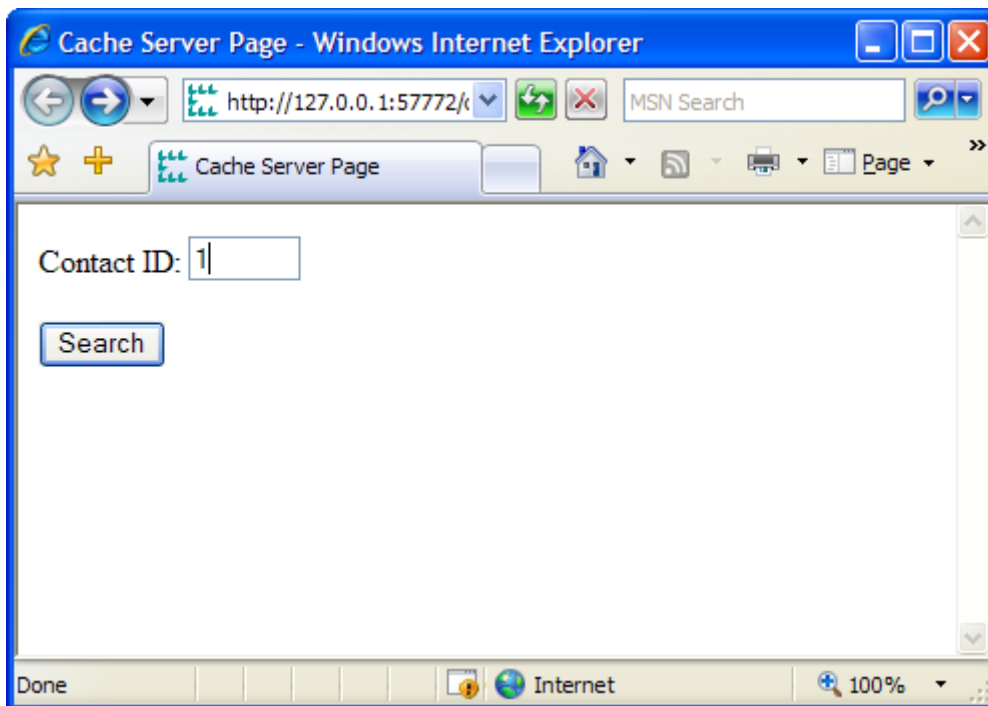
3. 上記のスクリプトの後に、以下の<csp:while>タグを追加します。このタグは、クエリの結果セットに繰り返し処理を行い、クエリの結果を表示します。

```
<csp:while condition="rs.Next()">  
<p>#(rs.Data("PhoneNumberType"))# #(rs.Data("Number"))#</p>  
</csp:while>
```

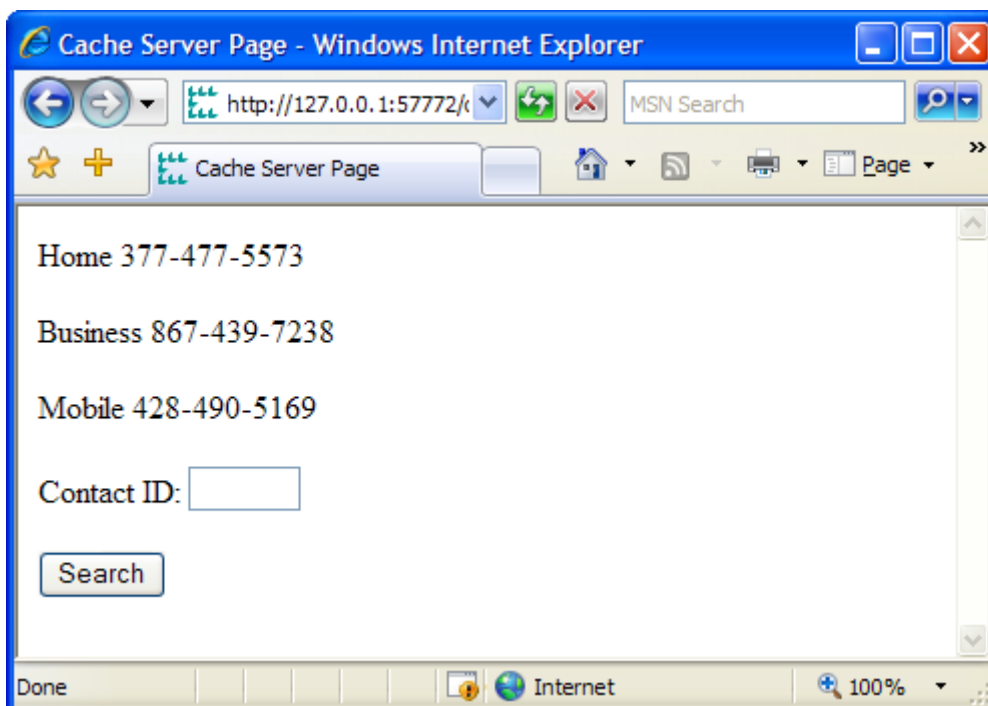
4. 最後に、MySamplePage2.CSP に以下のフォームを追加します。このフォームには、1 つのテキスト・フィールドと[送信]ボタンがあります。ユーザがこのボタンをクリックすると、フォーム・フィールドの値がページに送信されます。これがダイナミック・クエリで使用する検索値になります。

```
<form>  
<p>Contact ID: <input type="text" size="5" name="ID"/></p>  
<input type="SUBMIT" Name="Search" Value="Search" />  
</form>
```

5. Web ブラウザで MySamplePage2.CSP を開きます。テキスト・フィールドに有効な **Contact** ID 値を入力します。



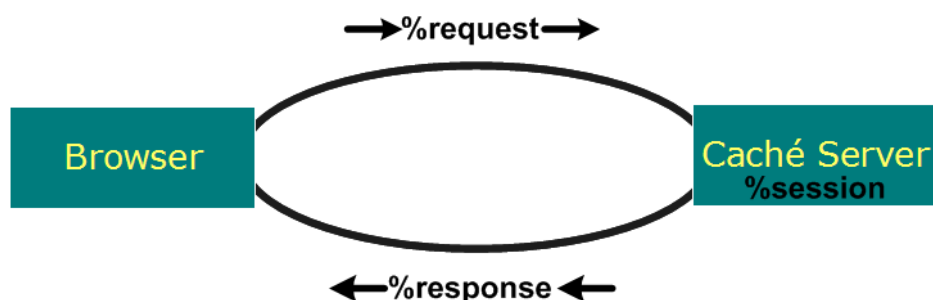
6. **[検索]** ボタンをクリックします。ID 値が 1 の **Contact** の電話番号がページに表示されます。



## 状態の管理: %CSP.Request、%CSP.Session、%CSP.Response

HTTP はステートレス・プロトコルであり、ページ要求間で情報を格納するためのリソースはありません。ただし、CSP にはページ要求間で情報を渡すための 3 種類のオブジェクトがあります。

- **[%CSP.Request](#)** — このクラスは、CSP ページに対する受信要求についての情報をカプセル化します。CSP サーバが要求を受信すると、要求に関する情報でクラスをインスタンス化します。サーバは、%request 変数によって要求を処理する CSP ページでこのオブジェクトを使用できるようにします。%request に含まれるデータには、ページ要求の URL のクエリ文字列に含まれる値があります。CSP サーバは要求を処理するページを生成した後、%request オブジェクトを破棄します。
- **[%CSP.Session](#)** — このクラスは、CSP アプリケーション内の各セッションに関する情報をカプセル化します。セッションは、ユーザが初めてアプリケーションのページを要求すると開始し、アプリケーションのさまざまなページを移動している間は継続します。一定の期間が経過するか(タイムアウト)、ユーザがアプリケーションを終了すると、セッションは終了します。セッションが開始すると、CSP サーバは **[%CSP.Session](#)** のインスタンスを作成し、CSP サーバにこのインスタンスを格納します。オブジェクトの名前は %session です。セッション内で要求を処理する各ページでこのオブジェクトを使用できます。アプリケーションは、セッションの異なるページ要求間で必要なユーザ識別などの情報を %session 内に格納できます。
- **[%CSP.Response](#)** — このクラスは、生成されるページの HTTP ヘッダのコンテンツに関する情報をカプセル化します。CSP サーバは %response という名前のオブジェクトをインスタンス化して、要求に回答する CSP ページでこのオブジェクトを使用できるようにします。%response を使用すると、CSP ページでは、例えば回答のコンテンツ・タイプを設定したり、要求を別の CSP ページにリダイレクトしたりすることができます。



## %CSP.Request

**[%CSP.Request](#)** クラスは、多数のプロパティとメソッドをサポートしています。以下で、その一部について説明します。

### %Request のプロパティとメソッド

プロパティまたはメソッド	説明
URL	要求の URL。クエリ文字列は含まれません。
Data	URL クエリ文字列の名前と値のペアを含む配列。指定された 1 つの名前に複数の値が関連付けられる場合、インデックス 1 で開始する添え字付きの値になります。 <ul style="list-style-type: none"> <li>• <code>%request.Data(param, 1)</code> を使用して、param に関連付けられている値を取得します。</li> </ul> <p>Caché Object Script の <code>\$Data</code> 関数を使用して、値が特定の名前に割り当てられているかどうかを確認できます。値が特定の名前に割り当てられていない場合は、<code>\$Get</code> を使用します。<code>\$Get</code> は、パラメータが定義されていない場合、“NULL” 文字列を返します。</p>
Cookies	要求に関連付けられた cookie の値を含む文字列の配列。
Count	Data の指定された名前に関連付けられた値の数を返します。
GetCookie	特定の cookie を返します。

#### Note:

**[%CSP.Request](#)** の詳細は、Caché クラスのドキュメントと、Caché ドキュメント: [\[Caché 開発ガイド\] - \[Caché Server Pages \(CSP\) の使用法\] - \[CSP での HTTP 要求\] - \[%CSP.Request オブジェクト\]](#) を参照してください。

Cookie の使用の詳細は、Caché ドキュメント: [\[Caché 開発ガイド\] - \[Caché Server Pages \(CSP\) の使用法\] - \[CSP セッション管理\] - \[Cookie へのデータの格納\]](#) を参照してください。

## %CSP.Request の例:クエリ文字列

クエリ文字列は、“?”記号で URL に付加される名前と値のペアです。例えば、クエリ文字列に名前と値のペア“OBJID=1”が付加された URL は以下ようになります。

```
http://127.0.0.1:57772/csp/user/Contact.csp?OBJID=1
```

CSP サーバは、クエリ文字列の名前と値のすべてのペアを、CSP ページに送信される%request オブジェクトの Data プロパティに配置します。以下のコードは、%request の Data プロパティから OBJID の値を取得します。

```
$Get(%request.Data("OBJID",1))
```

以下に、URL のクエリ文字列で渡されるオブジェクト ID を持つ **Contact** オブジェクトを結合する <csp:object>タグを示します。

```
<csp:object name="objForm" classname="CSPTutorial.Contact"  
OBJID=#($Get(%request.Data("OBJID",1)))#/>
```

## %CSP.Request の例:フォーム・フィールド

CSP ページの例 LogIn.csp には、以下のフォームが含まれます(LogIn.csp の場所については下記のメモを参照してください)。

```
<form name="edit" method="post" action="StateTest.CSP">
<p>User Name: <input type="text" name="txtUserName"></p>
<p>Password: <input type="text" name="txtPassword"></p>
<p><input type="submit" name="btnLogIn" value="LogIn" ></p>
</form>
```

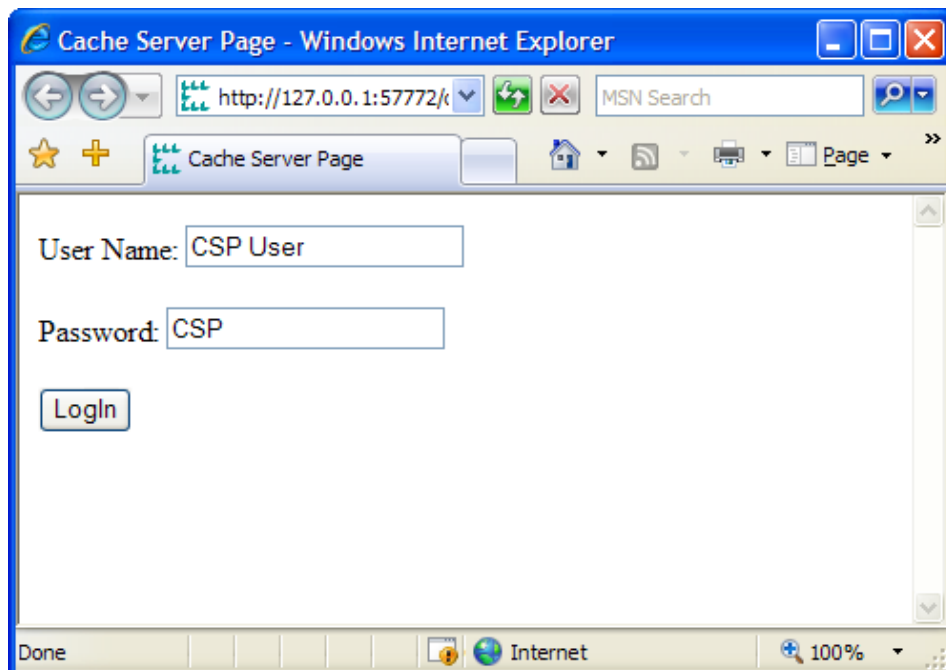
ユーザがこのフォームを選択した場合、つまり[送信]ボタンをクリックすると、ブラウザは StateTest.CSP を要求します。 ページに渡される%request 変数には、LogIn.CSP のフォーム・フィールドに入力した値が含まれます。

%request をテストするには、以下の手順を実行します。

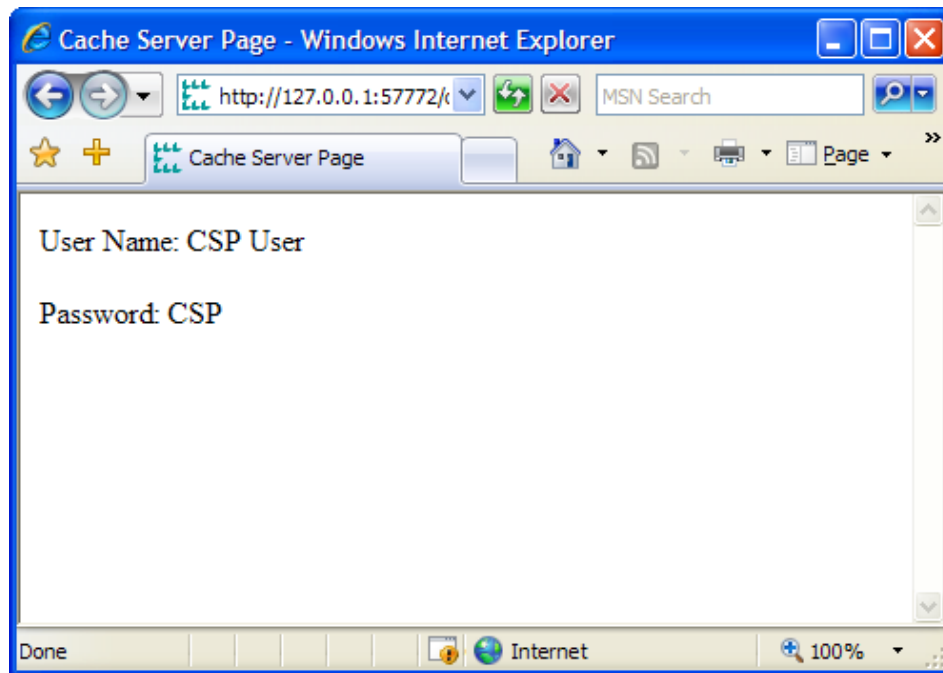
1. Caché スタジオを使用して、csp/user に StateTest.CSP を作成します。
2. 以下のコードを、StateTest.CSP の<BODY>セクションに追加します。 このコードは、%request に格納されている LogIn.CSP フォーム・フィールドの値を取得して表示します。

```
<p>User Name: #($Get(%request.Data("txtUserName",1)))#</p>
<p>Password: #($Get(%request.Data("txtPassword",1)))#</p>
```

3. ブラウザで LogIn.CSP を開きます。 フォーム・フィールドにテキストを入力します。



4. **[LogIn]**をクリックします。StateTest.CSP には、LogIn.CSP フォーム・フィールドに入力した値が表示されます。



**Note:**

LogIn.CSP は、チュートリアルに付随するサンプル・ファイルの 1 つです。サンプル・ファイルのインストールの詳細は、[Appendix A: チュートリアル・ファイルのインストール] を参照してください。

## %CSP.Session

**%CSP.Session** クラスは、多数のプロパティとメソッドをサポートしています。以下で、その一部について説明します。

### %Session のプロパティとメソッド

プロパティまたはメソッド	説明
SessionId	システムによってセッションに割り当てられる一意の <b>%String</b> 値。
Data	アプリケーションに固有の、名前と値のペアの配列。このプロパティを使用して、複数の要求で使用するセッション関連のデータを格納します。
AppTimeout	アプリケーションのタイムアウト値を秒単位で指定します。セッション内でページ要求をしている間に指定した時間が経過すると、セッションを閉じます。タイムアウトを指定しない場合、プロパティの値は 0 に設定されます。

#### Note:

**%CSP.Session** の詳細は、Cache クラスのドキュメントと、Cache ドキュメント: [\[Cache 開発ガイド\]](#) - [\[Cache Server Pages \(CSP\) の使用法\]](#) - [\[CSP セッション管理\]](#) - [\[%CSP.Session オブジェクト\]](#) を参照してください。

## %CSP.Session の例

CSP で `%session` を使用するには、以下の手順を実行します。

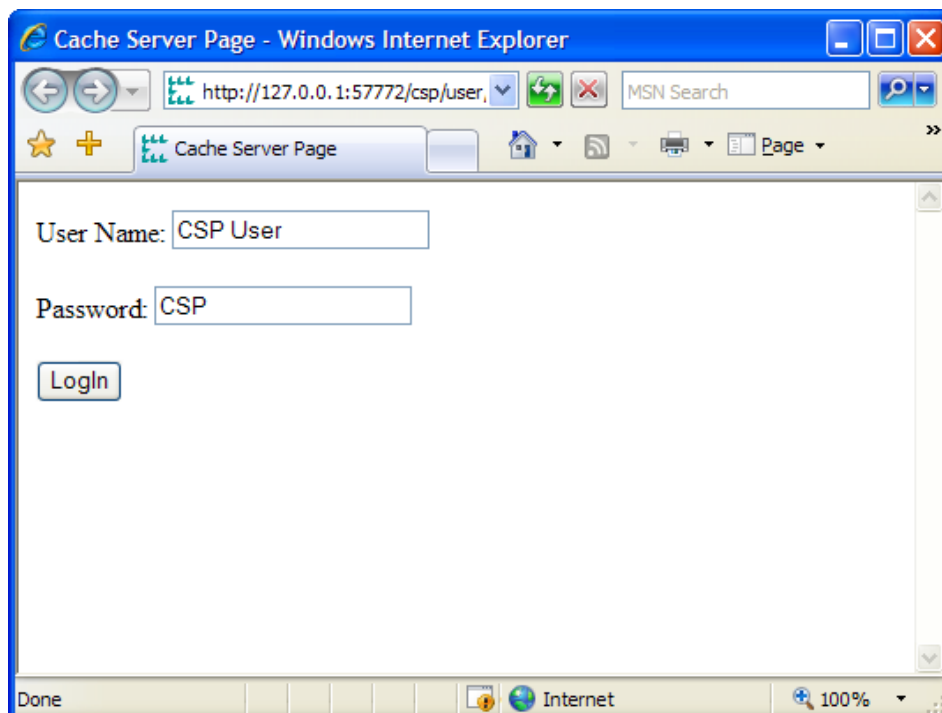
1. Caché スタジオで LogIn.CSP を開きます。以下の value 属性を追加して、フォームの入力コントロールを変更します。このコードは、`%session` に格納されている値をフォームのフィールドに表示します。

```
...  
<p>User Name: <input type="text" name="txtUserName"  
value="#($Get(%session.Data("Usr", 1)))#"></p>  
<p>Password: <input type="text" name="txtPassword"  
value="#($Get(%session.Data("Pwd", 1)))#"></p>  
...
```

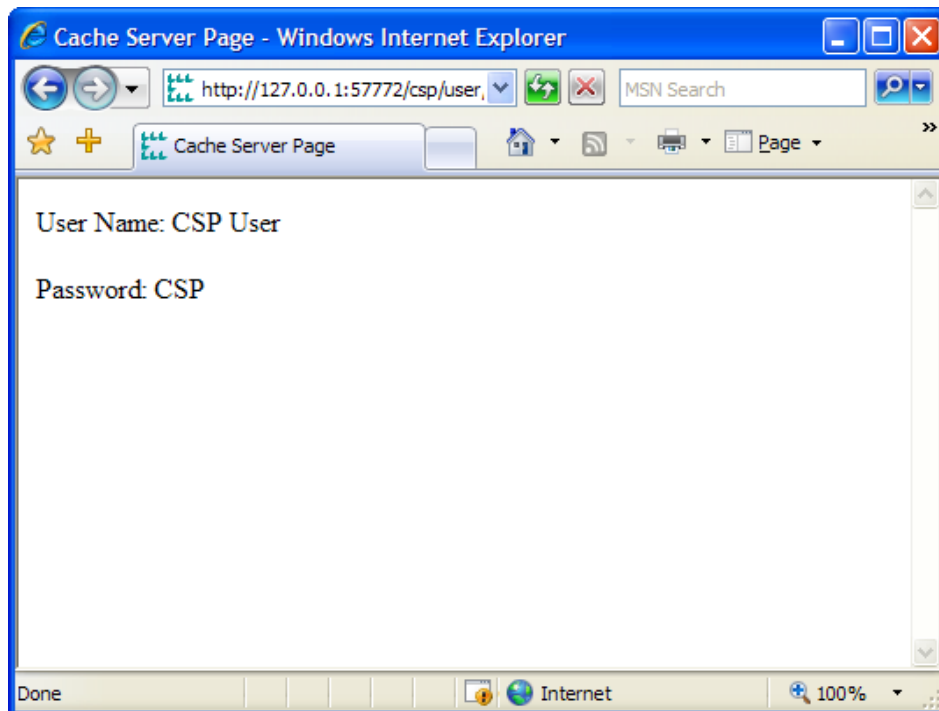
2. 以下のスクリプトを StateTest.CSP に追加します。このコードは、LogIn.CSP のフォーム・フィールドに入力した値を `%request` から取得し、これらの値を `%session.` に格納します。

```
...  
<script language="cache" runat="server">  
Set %session.Data("Usr", 1) = $Get(%request.Data("txtUserName", 1))  
Set %session.Data("Pwd", 1) = $Get(%request.Data("txtPassword", 1))  
</script>
```

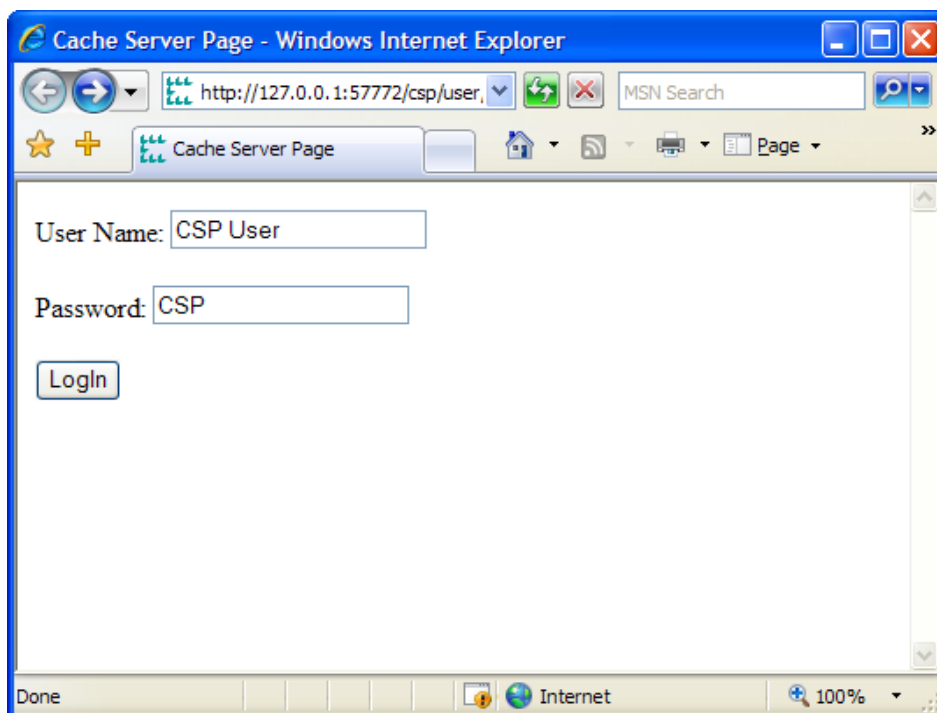
3. Web ブラウザで LogIn.CSP を開きます。フォームのフィールドに値は表示されていません。フォーム・フィールドにテキストを入力します。



4. **[LogIn]**をクリックします。StateTest.CSP が開き、LogIn.CSP のフォーム・フィールドに入力した値が表示されます。



5. LogIn.CSP に戻るには、ブラウザの "[戻る]" ボタンをクリックします。フォーム・フィールドには、最初にこのページにアクセスしたときに入力したテキストが表示されます。



**Note:**

LogIn.CSP は、チュートリアルに付随するサンプル・ファイルの 1 つです。 サンプル・ファイルのインストールの詳細は、[Appendix A: チュートリアル・ファイルのインストール] を参照してください。

## %CSP.Response

**%CSP.Response** クラスは、多数のプロパティとメソッドをサポートしています。以下で、その一部について説明します。

### %Response のプロパティとメソッド

プロパティまたはメソッド	説明
<code>Redirect</code>	このプロパティの値をターゲット URL に設定します。これにより、HTTP リダイレクト・コードがブラウザに返されます。ブラウザは、URL で指定されたページを要求します。
<code>ServerSideRedirect</code>	このプロパティの値をターゲット URL に設定します。URL にサーバ名は使用できません。例えば、/csp/user/Contact.csp のような URL になります。ServerSideRedirect を使用したリダイレクトは、CSP サーバ上で完全に行われます。ServerSideRedirect を使用したリダイレクトは、クライアントとの往復が 1 回分少ないため、Redirect を使用したリダイレクトよりも短時間で済みます。ServerSideRedirect の欠点の 1 つとして考えられることは、クライアント・ブラウザに表示される URL が、リダイレクト先のページではなく、最初に要求したページの URL になってしまう点です。この場合、ユーザは混乱する可能性があります。
<code>ContentType</code>	応答のコンテンツ・タイプを設定します。既定値は、 <code>text/html</code> です。
<code>Cookies</code>	応答で送信される cookie の配列。
<code>SetCookie</code>	応答に cookie を設定します。

これは HTTP ヘッダに影響を与えるため、`onPreHTTP` メソッドで `%response` 変数を操作します。

#### Note:

**%CSP.Response** の詳細は、Caché クラスのドキュメントと、Caché ドキュメント: [\[Caché 開発ガイド\]](#) - [\[Caché Server Pages \(CSP\)の使用方法\]](#) - [\[CSP での HTTP 要求\]](#) - [\[%CSP.Response オブジェクトおよび ONPreHTTP メソッド\]](#) を参照してください。

Cookie の使用の詳細は、Caché ドキュメント: [\[Caché 開発ガイド\]](#) - [\[Caché Server Pages \(CSP\)の使用方法\]](#) - [\[CSPセッション管理\]](#) - [\[Cookieへのデータの格納\]](#) を参照してください。

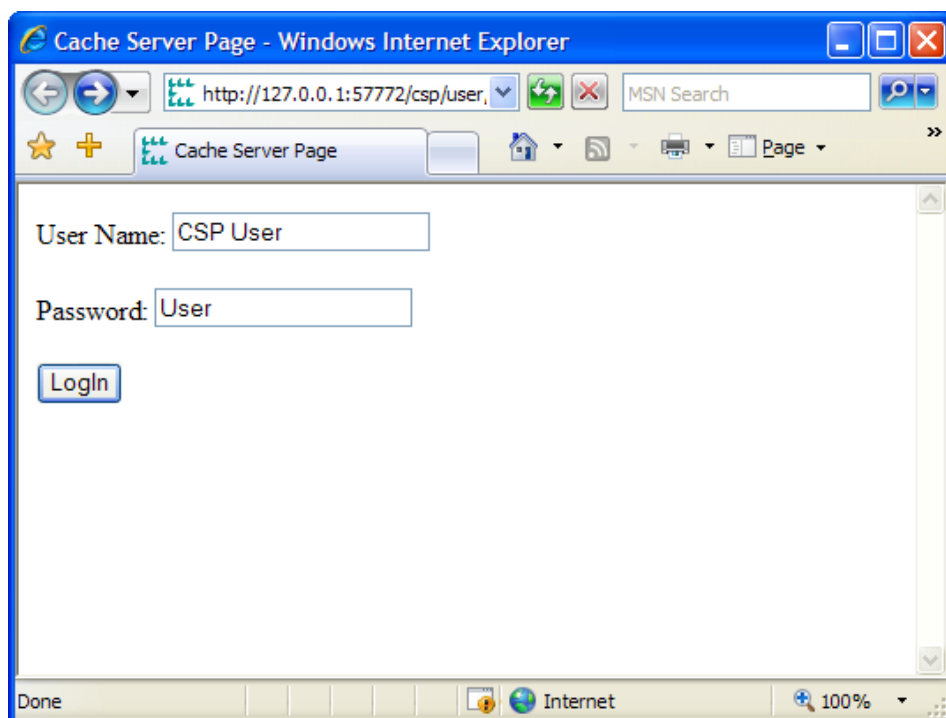
## %Response の例:リダイレクト

`%response` とその `Redirect` プロパティでリダイレクトを使用するには、以下の手順を実行します。

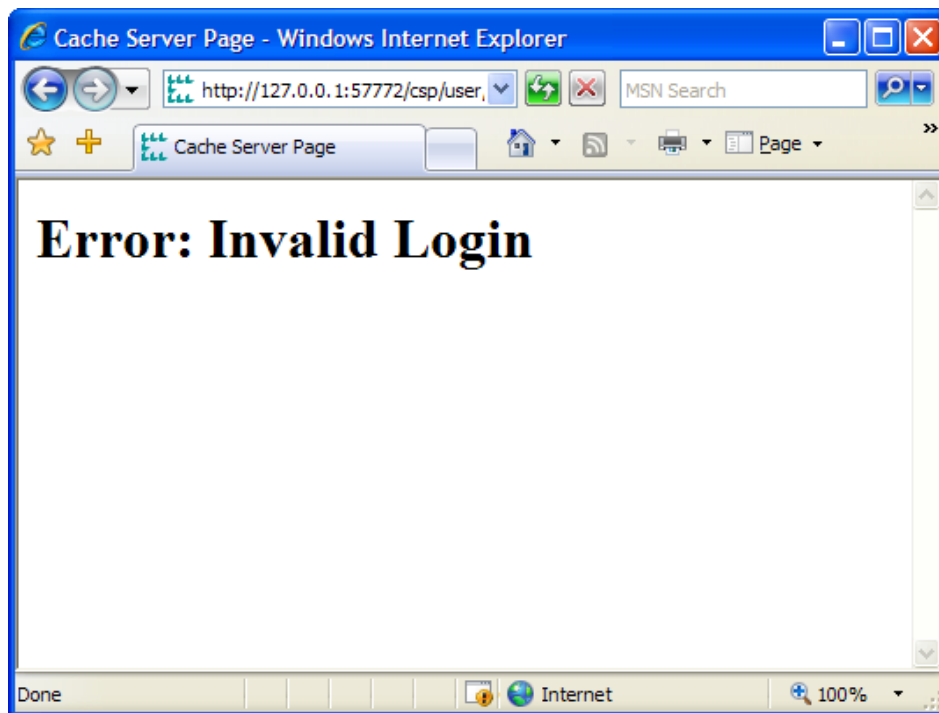
1. 既存のコードを変更せずに、`StateTest.csp` の `<Body>` セクションに以下のスクリプトを追加します。このスクリプトは、ページの `onPreHTTP` メソッドを定義します。このメソッドは、`LogIn.CSP` に入力したパスワードの値をテストします。値が `"CSP"` ではない場合、メソッドはユーザを `Error.CSP` にリダイレクトします(このファイルの場所については下記のメモを参照してください)。それ以外の場合、`StateTest.CSP` がロードされます。`onPreHTTP` は 1 を返します。0 を返した場合、`StateTest.CSP` はロードされていません。

```
<script language="cache" method="OnPreHTTP" arguments=""
returntype="%Boolean">
if ($Get(%request.Data("txtPassword",1))' = "CSP")
{
set %response.Redirect="Error.CSP"
}
quit 1
</script>
```

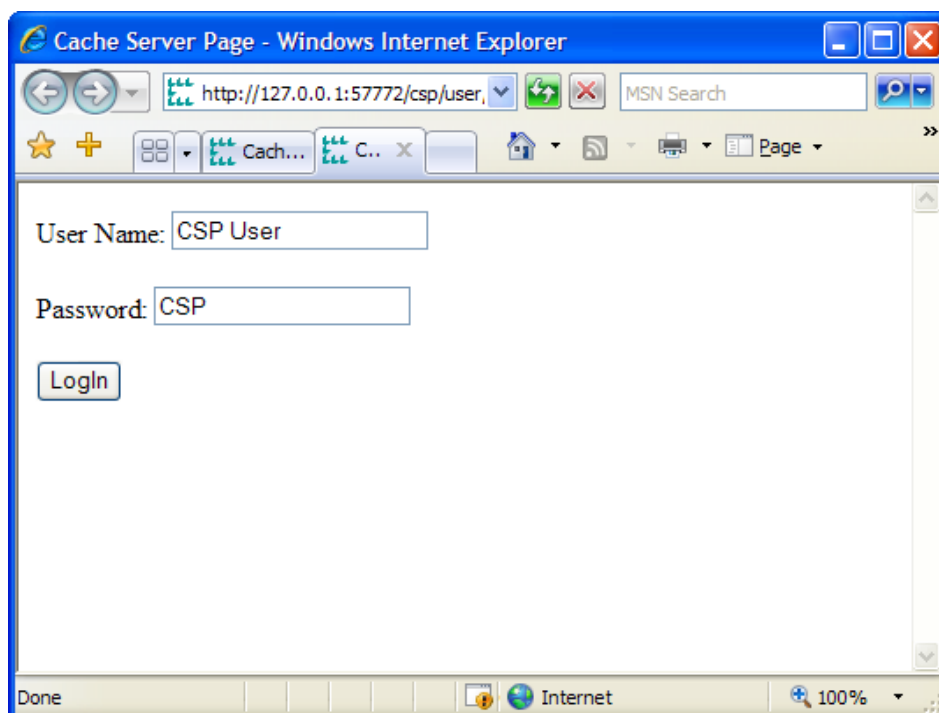
2. Web ブラウザで `LogIn.CSP` を開きます。パスワード・フィールドに `"CSP"` 以外のテキストを入力します。



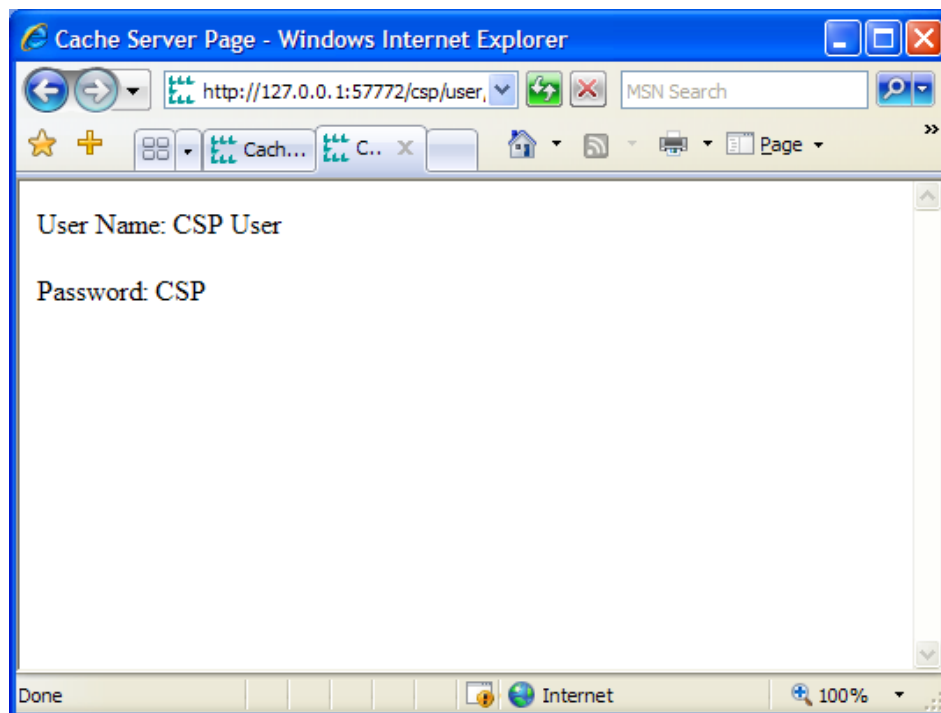
3. **[LogIn]**をクリックします。ブラウザは Error.CSP にリダイレクトされます。



4. Web ブラウザで LogIn.CSP を開きます。パスワード・フィールドに“CSP” と入力します。



5. **[LogIn]**をクリックします。ブラウザに StateTest.CSP がロードされます。



**Note:**

Error.CSP はチュートリアルに付随するサンプル・ファイルの 1 つです。 サンプル・ファイルのインストールの詳細は、[Appendix A: チュートリアル・ファイルのインストール]を参照してください。

## ハイパーイベント

ハイパーイベントは、CSP ページの JavaScript イベント処理コードが Caché サーバで Caché メソッドを実行できる CSP 機能です。ハイパーイベントを使用すると、Caché サーバのメソッドは、ページを再ロードせずにページに表示された情報を更新できます。さらに、Caché サーバのメソッド自体が、クライアント・ブラウザに返されるとすぐに実行する JavaScript コードを返すことができます。

ハイパーイベントを使用するための構文オプションは以下のとおりです。

### ハイパーイベント構文

構文	説明
<pre>#server(package.class.method(JS args))# #server(..method(JS args))# #vserver(package.class.method(JS args))# #vserver(..method(JS args))#</pre>	<ul style="list-style-type: none"> <li>• #server()#を使用して Caché ObjectScript メソッドを実行します。</li> <li>• #vserver()#を使用して Caché Basic メソッドを実行します。</li> <li>• method は任意のクラス・メソッドです。</li> <li>• JS args は、JavaScript 変数または DOM 参照、あるいは JavaScript 変数や DOM 参照を返す CSP 式です。</li> </ul>
<pre>#call(package.class.method(JS args))# #call(..method(JS args))#</pre>	<p>このメカニズムと上記のメカニズムは大幅に異なります。</p> <ul style="list-style-type: none"> <li>• メソッド呼び出しは非同期です。</li> <li>• メソッド呼び出しはクライアントに値を返すことができません。</li> </ul>

ハイパーイベントの実装には、3 つの異なるオプションがあります。CSP アプリケーションを構成するときに実装を選択します。クライアント・ブラウザがその機能に基づいて実装を選択するように、アプリケーションを構成できます。

## ハイパーイベントの実装

実装オプション	説明
XMLHttpRequest オブジェクト	推奨される実装です。ほとんどのブラウザと互換性があります。暗号化された HTTP 要求を CSP サーバに送信します。
CSP イベント・ブローカー	クライアント・ブラウザにインストールする必要がある Java アプレットです。暗号化された HTTP 要求を CSP サーバに送信します。すべてのブラウザと互換性があるわけではありません。
IFRAME タグまたは LAYER タグ	#call 指示文でのみ使用します。すべてのブラウザと互換性があります。

## ハイパーイベントの例

ハイパーイベントおよび#server()#指示文を使用するには、以下の手順を実行します。

1. Caché スタジオを使用して、csp/user に MySamplePage3.CSP という名前の CSP ページを作成します。
2. ページに以下のスクリプトを追加します。このスクリプトは、**AlertUser** という名前の Caché メソッドを定義します。このスクリプトは埋め込み JavaScript 構文<js></js>を使用する点に注意してください。CSP コンパイラは、埋め込み JavaScript を、Caché ObjectScript の適切な **Write** 文のセットに自動的に変換します。

```
<script language="cache" method="AlertUser">  
&js<alert("Hello User");>  
</script>
```

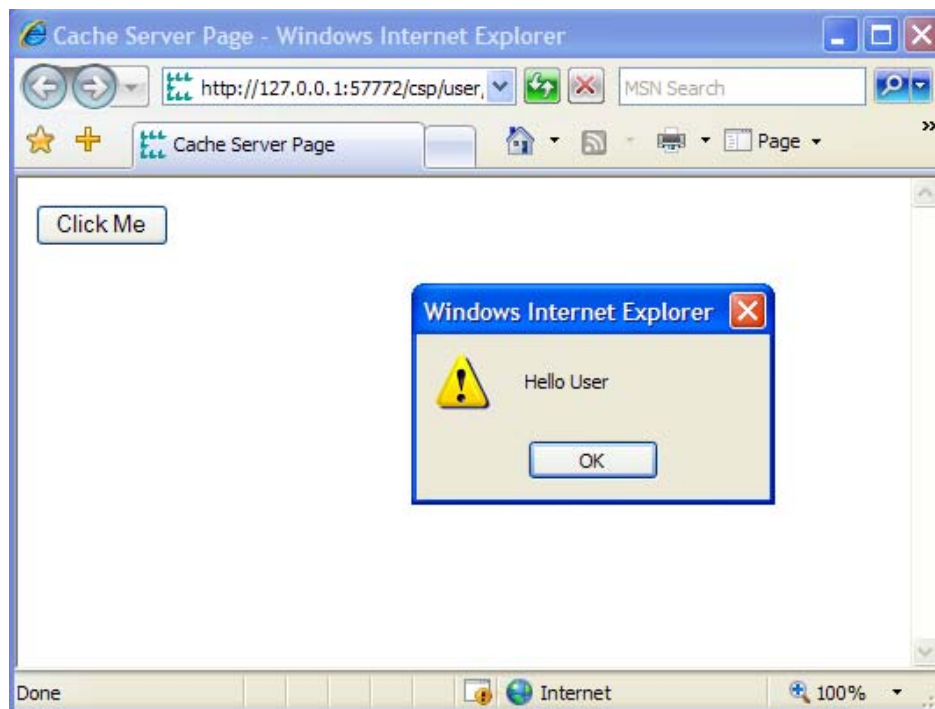
3. 同様に以下のスクリプトを追加します。このスクリプトは、**MyFunction** という名前の JavaScript 関数を定義します。この関数は、#server()#を使用して、サーバ側の **AlertUser** メソッドを呼び出します。

```
<script language="JavaScript">  
function MyFunction() {  
#server(..AlertUser())#  
}  
</script>
```

4. 最後に、以下のボタンを MySamplePage3.CSP に追加します。ボタンをクリックすると、JavaScript 関数 **MyFunction** が呼び出されます。

```
<input type="Button" value="Click Me" onClick="MyFunction()"/>
```

5. Web ブラウザで MySamplePage3.CSP を開き、このボタンをクリックします。



**Note:**

埋め込みJavaScriptと<script>の詳細は、Cacheドキュメント: [\[Cache 開発ガイド\]](#) - [\[Cache Server Pages \(CSP\)の使用法\]](#) - [\[CSP におけるタグを使用した開発\]](#) - [\[サーバからのクライアント・イベントへの応答\]](#) に関する説明を参照してください。

HTML の埋め込みに<script>を使用することもできます。

## ハイパーイベントと DOM

HTML Document Object Model(DOM)は、プラットフォームや言語に依存しない、HTMLドキュメントの要素のインタフェースを定義します。DOMを使用すると、ハイパーイベントや JavaScript は、HTMLドキュメントのコンテンツ、スタイル、構造にアクセスしてこれらを動的に更新できます。DOMは階層構造であり、要素内に要素のツリー構造を定義します。特定の要素にアクセスするには、最上位要素から中間要素を経由して目的の要素を参照します。名前(HTML要素の `name` 属性の値)や、さまざまな要素配列における要素の位置によって、HTML要素を参照できます。

- `self.document` — ツリーの最上位ノード。ここから参照が始まります。“`self`”が必要なブラウザと不要なブラウザがあります。
- `self.document.forms.MyForm` — `MyForm` という名前のドキュメントのフォーム。
- `self.document.forms.MyForm.Field1` — `MyForm` というフォームの `Field1` という入力フィールド。
- `self.document.forms.MyForm.Field1.value` — `MyForm` というフォームの `Field1` という入力フィールドの現在の値。

### Note:

HTML DOMの詳細は、

「[W3 Schools](http://www.w3schools.com/default.asp)」Webサイト(<http://www.w3schools.com/default.asp>)の

「[HTML DOM Tutorial](http://www.w3schools.com/html/dom/default.asp)」(<http://www.w3schools.com/html/dom/default.asp>)を参照してください。

## ハイパーイベントと DOM の例

以下の例では、ハイパーイベントと DOM を使用して、ハイパーイベントの LogIn ページを作成します。JavaScript イベント・ハンドラ・コードは、Cache ObjectScript イベント・ハンドラを呼び出します。これは HTML DOM を使用して、フォームからユーザーが入力したログイン情報を取得します。また、HTML DOM を使用して、ユーザーのブラウザを正しい CSP ページに導きます。ハイパーイベントの LogIn ページを作成およびテストするには、以下の手順を実行します。

1. Caché スタジオを使用して、csp/user に、HyperLogIn.CSP および Welcome.CSP という 2 つの新しい CSP ページを作成します。
2. Welcome.CSP の<body></body>タグ間によるこそメッセージを以下のように追加します。

```
<body>
<h1>Welcome!</h1>
</body>
```

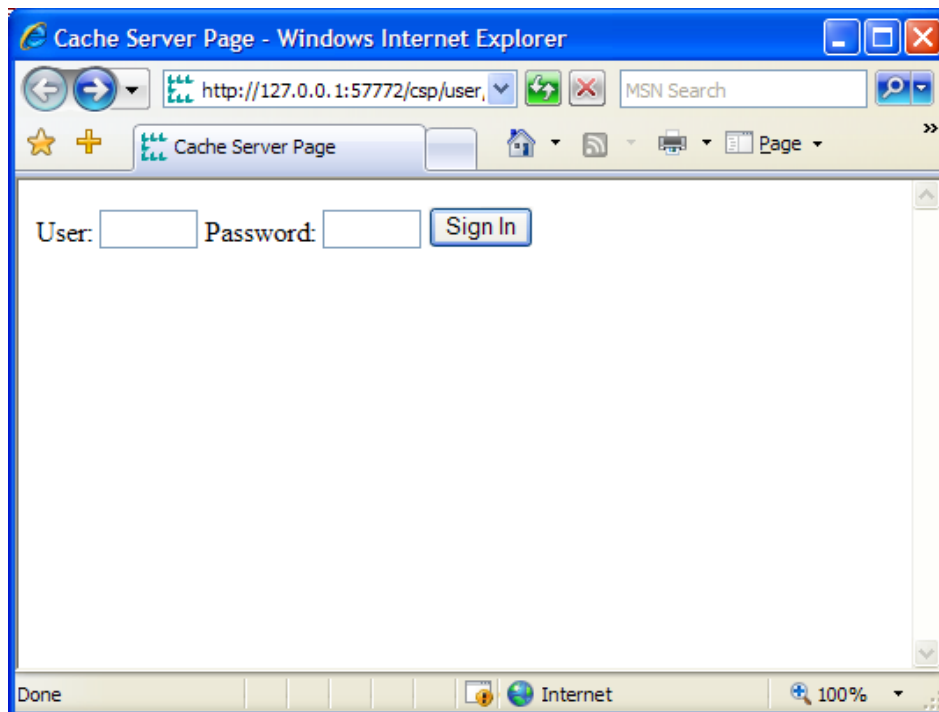
3. 以下のスクリプトを HyperLogIn.CSP に追加します。これは Caché ObjectScript の検証機能を定義します。ユーザーがフォームのパスワード・フィールドに“CSP”と入力した場合、Welcome.CSP をロードするようにユーザーのブラウザに指示します。ユーザーが“CSP”以外のテキストを入力した場合、Error.CSP をロードするようにユーザーのブラウザに指示します。

```
<script language="cache" method="Validate"
  arguments="Pwd:String" returnType="%Boolean">
  if (Pwd = "CSP") {
    &js<self.document.location=' Welcome.CSP' >
  }
  else {
    &js<self.document.location=' Error.CSP' >
  }
  quit 1
</script>
```

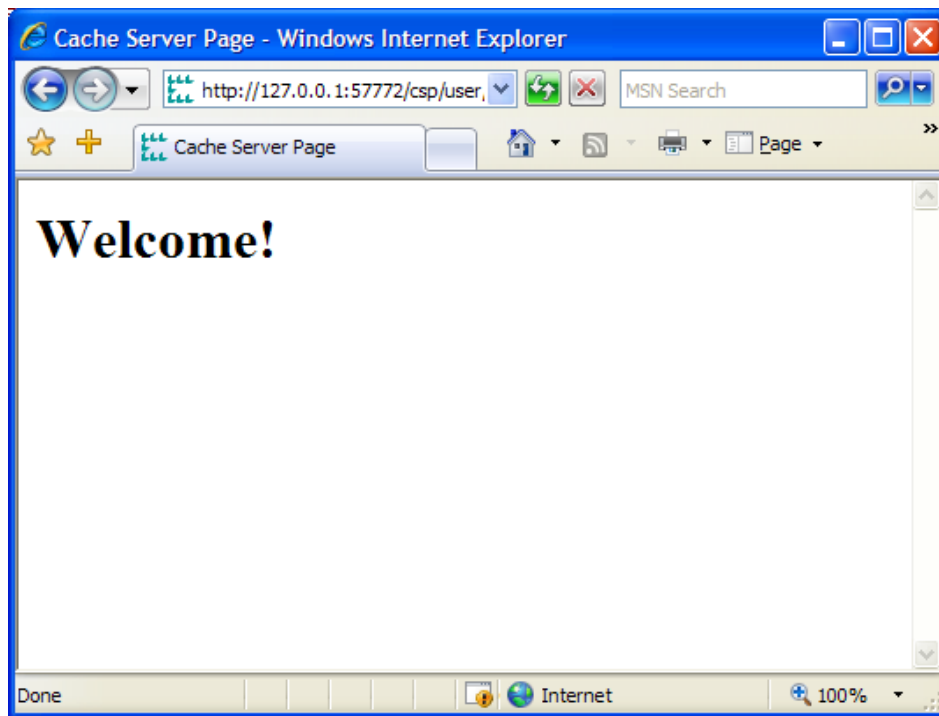
4. 以下のフォームを追加します。ボタンの `onClick` 属性の値は、ハイパーイベントを使用して **Validate** メソッドを呼び出す JavaScript イベント・ハンドラです。

```
<form name="form1" method="post" action="">
  User: <input type="text" name="txtUsr" size="5">
  Password: <input type="password" name="txtPwd" size="5">
  <input type="button" value="Sign In"
  onClick="#server(..Validate(self.document.form1.txtPwd.value))#">
</form>
```

5. Web ブラウザで HyperLogIn.CSP を開きます。パスワード・フィールドに "CSP" と入力します。



6. **[Sign In]**をクリックします。



## 演習

次の演習を通して、CSP Contact Management システムの作成を学習します。アプリケーションは以下の 3 つの CSP ページから構成されています。

1. Contact.CSP — 各連絡先のすべての情報（電話番号など）を表示します。ユーザは連絡先を編集できますが、電話番号情報は編集できません。
2. Phone.CSP — 各電話番号のすべての情報を表示します。ユーザは情報を編集できます。
3. AddPhoneNumber.CSP — 新しい電話番号を作成し、これらを連絡先に追加します。

この演習を実行する前に、作業するネームスペースに **Contact** および **PhoneNumber** クラスをインストールして生成します。手順については、[Appendix A: チュートリアル・ファイルのインストール] を参照してください。

演習 1: Contact.CSP: Contact.CSP という名前の CSP ページを作成します。このページを使用すると、ユーザは Contact Information を表示および更新できます。このページはチュートリアルの第 2 章の例で部分的に作成している点に注意してください。Contact.CSP は以下の要件を満たす必要があります。

1. **Contact** クラスに結合されているフォームが含まれること。このフォームには、Name、ContactType、**Contact** インスタンスの ID プロパティが表示されます。ユーザは Name および ContactType プロパティを更新できます。Caché スタジオのフォーム・ウィザードを使用してこのフォームを作成できます。このチュートリアルの第 2 章 / Chapter 7: の「フォーム・ウィザードの例」セクションには、このフォームの作成に関する演習ガイドがあります。
2. フォームの“検索”ページで、ContactType を基準にした検索が可能であること。検索結果の Name および ContactType の両方が表示されます。
3. フォームで開いている **Contact** インスタンスに属している **PhoneNumber** インスタンスごとに、Number および PhoneNumberType がページに表示されること。電話番号は、ダイナミック・クエリまたはクラス・クエリによって、データベースから取得されます。電話番号表示は、フォームの一部ではありません。
4. ページに、Phone.CSP(演習 2 で作成したページ)とリンクする各電話番号がハイパーリンクで表示されること。
5. 各ハイパーリンクの URL にクエリ文字列が含まれ、そのクエリ文字列に、**PhoneNumber** インスタンスの ID が OBJID の値として含まれること。

演習 2: Phone.CSP:Phone.CSP という名前の CSP ページを作成します。このページを使用すると、ユーザは PhoneNumber Information を表示および更新できます。Phone.CSP は以下の要件を満たす必要があります。

1. **PhoneNumber** クラスに結合されているフォームが含まれること。フォームには、Numer、PhoneNumberType、ID、および関連付けられた **Contact** の Name が表示されます。ユーザは PhoneNumberType および Number プロパティを編集できます。Caché Web フォーム・ウィザードを使用してこのフォームを作成できます。
2. Contact.csp にリンクするハイパーリンクが含まれること。ハイパーリンク URL にクエリ文字列が含まれ、そのクエリ文字列に、OBJID の正しい **Contact** ID 値が含まれること。

演習 3: AddPhoneNumber.CSPAddPhoneNumber.CSP という名前の CSP ページを作成します。このページを使用すると、ユーザは新しい電話番号を連絡先に追加できます。AddPhoneNumber.CSP は以下の要件を満たす必要があります。

1. ユーザが新しい **PhoneNumber** の PhoneNumberType および Number プロパティを指定するためのフォームがあること。このフォームには、保存ルーチンを有効化する “[保存]” ボタンもあります。このフォームをオブジェクトに結合するべきではありません。Web フォーム・ウィザードを使用してこのフォームを作成することはできません。
2. 新しい **PhoneNumber** インスタンスが、Caché ObjectScript または Caché Basic 関数で作成および保存されること。
3. “[保存]” ボタンは、ハイパーイベントを使用して、新しい **PhoneNumber** を作成する関数を有効にします。**PhoneNumber** のプロパティ値は、HTML DOM を使用してフォームから取得されます。
4. 新しい **PhoneNumber** を作成する関数が、**PhoneNumber** Contact プロパティに、正しい **Contact** ID 値を割り当てること。
5. ページに Contact.CSP にリンクするハイパーリンクが含まれること。ハイパーリンク URL にクエリ文字列が含まれ、そのクエリ文字列に、正しい **Contact** ID が OBJID の値として含まれること。

**Note:**

PartIIExerciseSoln.xml には、演習の解答ファイルがあります。このファイルの場所と解答のインストールの手順については、[Appendix A: チュートリアル・ファイルのインストール] を参照してください。

## 要約

このチュートリアル第 2 章では、以下について学習しました。

- ページ指示文を使用したページ言語の設定。
- CSP ページおよび HTML フォームへのオブジェクトの結合。
- Caché スタジオのウィザードを使用した、Caché オブジェクトに結合される HTML フォームの作成。
- Caché オブジェクトを検索するための、カスタマイズしたページの作成。
- CSP ページへのクエリの追加。
- CSP アプリケーションの状態の管理。
- CSP ハイパーイベントを使用した、CSP ページからのサーバ側メソッドの呼び出し。

## チュートリアル・ファイルのインストール

標準の Caché インストールには、チュートリアルの演習や例を収めた、Caché クラスおよび CSP ページを含む XML ファイルのセットがあります。このファイルは

<cachsys>¥Dev¥Tutorials¥CSP にあります。

<http://vista.intersystems.com/samples/CSPTutorial.zip>からダウンロードすることもできます。

クラスと CSP ページをインストールするには、Caché スタジオを使用して Caché にクラスと CSP ページをインポートします。

1. Caché スタジオを開き、作業対象のネームスペース（ほとんどの場合 *USER*）に接続します。**[ファイル]→[ネームスペース変更]** をクリックして、目的のネームスペースに接続します。
2. Caché スタジオのメニュー・バーで、**[ツール]→[ローカルからインポート]** をクリックします。正しいファイルを参照します。目的のファイルをクリックし、**[開く]** をクリックします。
3. **[インポート]** ダイアログで、使用可能なすべての項目が選択され、**[インポートした項目をプロジェクトに追加する]** と **[インポートした項目をコンパイルする]** の両方のチェックが付けられていることを確認します。**[OK]** をクリックします。Caché は項目をインポートしてコンパイルします。
4. **CSPTutorial.Contact** (チュートリアルの第 I 章)、または **CSPTutorial.Contact** および **CSPTutorial.PhoneNumber** (チュートリアルの第 II 章) をサンプル・データで生成します。手順については、[Appendix B: 連絡先と電話番号の生成] を参照してください。

次に、チュートリアルの各章に付随するファイルについて説明します。

### 第 I 章のファイル

- *PartIStarter.xml* — **CSPTutorial.Contact** のクラス定義があります。第 I 章の例と演習を実行するには、このファイルをインストールして、クラスを生成します。
- *PartIExerciseSoln.xml* — 第 I 章の演習 (*Exercise1.CSP* および *Exercise2.CSP*) の解答があります。

### 第 II 章のファイル

- *PartIIStarter.xml* — **CSPTutorial.Contact** および **CSPTutorial.PhoneNumber** クラスと、*LogIn.CSP* および *Error.CSP* の CSP ページがあります。第 II 章の例と演習を実行するには、これらのファイルをインストールしてクラスを生成します。

## 連絡先と電話番号の生成

**CSPTutorial.Contact**、または **CSPTutorial.Contact** および **PhoneNumber** のクラス定義を含む XML ファイルをインポートしたら、Cache にクラスのサンプル・インスタンスを投入できます。手順は以下のとおりです。

1. クラスをコンパイルします。これを行うには、Cache スタジオで **[ビルド]**→**[コンパイル]** をクリックします。
2. クラスをインストールおよびコンパイルしたネームスペースで Cache ターミナルを開きます。Cache ターミナルが正しいネームスペースで開かない場合は、**ZN** コマンドを使用してネームスペースを変更します。例えば、*SAMPLES* でターミナルを開いており、クラスが *USER* にインストールされている場合は、次のように入力します。

```
SAMPLES>ZN "USER"  
  
USER>
```

3. **Contact** を生成するには、**Populate** メソッドを呼び出します。1 つの整数の引数で、作成するインスタンス数を指定します。親子リレーションシップがあるため、**PhoneNumber** を生成する前に **Contact** を生成する必要があります。

```
USER>Do ##class(CSPTutorial.Contact).Populate(5)  
  
USER>
```

4. **PhoneNumber** を生成する場合も、同じ手順を実行します。

```
USER>Do ##class(CSPTutorial.PhoneNumber).Populate(15)  
  
USER>
```

### Note:

**CSPTutorial.PhoneNumber** を生成する前に、**CSPTutorial.Contact** を生成する必要があります。これは 2 つのクラス間にリレーションシップがあるためです。

Cache データ生成ユーティリティの詳細は、Cacheドキュメント:[\[Cache開発ガイド\]-\[Cacheオブジェクトの使用法\]](#)-[\[Cache データ生成ユーティリティ\]](#) を参照してください。