

## 序文

Cache クイックスタート・チュートリアルへようこそ。

このチュートリアルは、Cache および Cache スタジオ開発環境を使用してダイナミックな Web 対応データベース・アプリケーションを迅速かつ簡単に構築する方法について説明します。

データベース設計とアクセスに関する、オブジェクト指向プログラミングのパワーと簡潔さを拡張するための多次元データベース・エンジンである Cache の使用方法について学習します。さらに、データにアクセスするダイナミック Web ページを迅速に作成するため、Cache Server Page(CSP)の使用方法についても学習します。

チュートリアルの学習を終えると、以下のことができるようになります。

- アプリケーション・データを表現する Cache クラスのコーディング
- Cache データへのダイナミックな Web ベースのアクセスを提供する CSP ページの作成
- Cache 組み込み自動生成ユーティリティを使用したデータベースのサンプル・データの生成
- アプリケーション内の Cache 組み込みクラスとユーティリティの活用
- Cache スタジオ、システム管理ポータル、および Cache ターミナルなどのさまざまな Cache 開発ツールの使用

このチュートリアルでは、主に完全に動作する Web 対応の Contact Management アプリケーションの作成について説明します。チュートリアルおよび演習では、アプリケーションのデータ・モデルを構築し、サンプル・データを取り込み、Web ベースのユーザ・インタフェースを作成する手順について説明します。

### Note:

< cachesys > ¥ dev ¥ tutorials ディレクトリには、作成済み Contact Management アプリケーションが含まれています。詳細は、Appendix A: [サンプル・アプリケーション] を参照してください。Windows で Cache を標準インストールしている場合、< cachesys > は C: ¥ InterSystems ¥ Cache になることに注意してください。Unix または Linux の環境では、< cachesys > は、/user/cachesys になります。

## Chapter1

## Caché ツールとテクノロジー

このチュートリアルでは、標準 Caché インストールで提供される迅速なアプリケーション開発ツールとテクノロジーの使用方法を学習します。

これらのツールとテクノロジーには、以下のものが含まれています。

- Caché スタジオ：アプリケーションの設計と構築に対するオブジェクト指向アプローチをサポートする統合開発環境。スタジオでは、Caché ObjectScript や Caché Basic などの複数の言語を使用してクラスを定義できます。
- Caché ターミナル：Caché データベースに対するコマンド行インターフェースを提供するウィンドウ。ターミナルによりオブジェクトをオープンしてメソッドを実行し、データを表示することができます。また、複数のコマンドを実行して、自動生成ユーティリティを含む複数の Caché ユーティリティにアクセスすることができます。
- Caché Server Pages (CSP)：ダイナミック Web ページを作成するためのテクノロジー。Caché スタジオを使用して CSP ページを作成することができます。
- システム管理ポータル：システムを管理するためのブラウザベースのインターフェース。このポータルを使用すると、データの定義、表示、アクセスなどの数多くのタスクをリレーショナル形式で実行できます。データベース・テーブルの定義やデータの照会は、標準の SQL を使用して実行できます。

**Note:**

Caché スタジオ、Caché ターミナル、およびシステム管理ポータルは、すべて Caché キューブから起動することができます。キューブをクリックしてから、メニューで各ツールをクリックします。

## Chapter2

## 統一データ・アーキテクチャ

Cache 多次元データベース・エンジンは、Cache データのオブジェクト指向およびリレーショナル表現の両方を提供する統一データ・アーキテクチャ(UDA)を作成します。このため、Cache データを、プロパティとメソッドを持つ Cache オブジェクトとしても、行、列、およびストアド・プロシージャを持つ SQL テーブルとしても扱うことができます。

このテーブルは、Cache オブジェクトの基本的なオブジェクト指向機能と、SQL テーブルのリレーショナル機能との間のマッピングを示しています。

## 統一データ・アーキテクチャ

Cache オブジェクト	SQL テーブル
パッケージ	スキーマ
クラス	テーブル
オブジェクト・インスタンス	テーブル行
プロパティ	テーブル列
メソッド	ストアド・プロシージャ
リレーションシップ	外部キー
埋め込みオブジェクト	列サブセット

このチュートリアルでは、アプリケーション・データのモデル化とアクセスのためのオブジェクト指向アプローチの使用 방법에重点を置いて説明します。ただし、同じデータにリレーショナル技術と SQL を使用してアクセスする方法についても学習します。

**Note:**

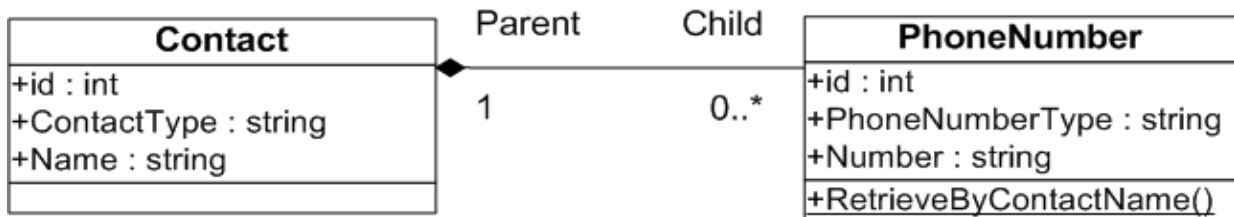
Cache統一データ・アーキテクチャの詳細は、Cacheドキュメント:[はじめに] - [Cache 入門] - [オブジェクト、SQL、および統一データ・アーキテクチャ] - [\[統一データ・ディクショナリ\]](#) を参照してください。

## Chapter3

## アプリケーション・データ・モデルおよびオブジェクト ID

Contact Management アプリケーションのデータ・モデルは、**Contact** と **PhoneNumber** の 2 つのクラスで構成されています。それぞれのクラスはデータベース内の個別のテーブルに対応します。クラスのインスタンスはそれぞれのテーブル内の行に相当します。2 つのクラスは関連しています。各 **Contact** インスタンスは複数の **PhoneNumber** インスタンスを含むことができます。例えば、1 つの **Contact** は、自宅、会社、および携帯電話の電話番号を表す複数の **PhoneNumber** インスタンスを含むことがあります。ただし、それぞれの **PhoneNumber** インスタンスは 1 つの **Contact** 内に含まれる必要があります。このリレーションシップは親子リレーションシップと呼ばれ、**Contact** が親で、**PhoneNumber** が子です。リレーションナルの観点から、**PhoneNumber** テーブルには、各行を **Contact** テーブルの行にリンクする外部キーが含まれます。

以下の UML 図は、**Contact** と **PhoneNumber** とのリレーションシップとそのプロパティを示します。



図に関してさらにいくつかの注意事項があります。

まず、両方のクラスに id プロパティが含まれます。どちらのクラスでも、このプロパティはオブジェクト ID を保持します。オブジェクト ID は、クラスのインスタンスを一意に識別します。または、リレーションナルの観点から、オブジェクト ID は、データベース・テーブルの行を一意に識別します。オブジェクト ID は、データベースに格納されているクラスのインスタンスをオープンするために使用することができます。Caché は自動的にオブジェクト ID をオブジェクトに割り当てます。

次に、idに加えて、それぞれのクラスは他の 2 つのプロパティを含みます。これらのプロパティをクラス定義に明示的に追加します。

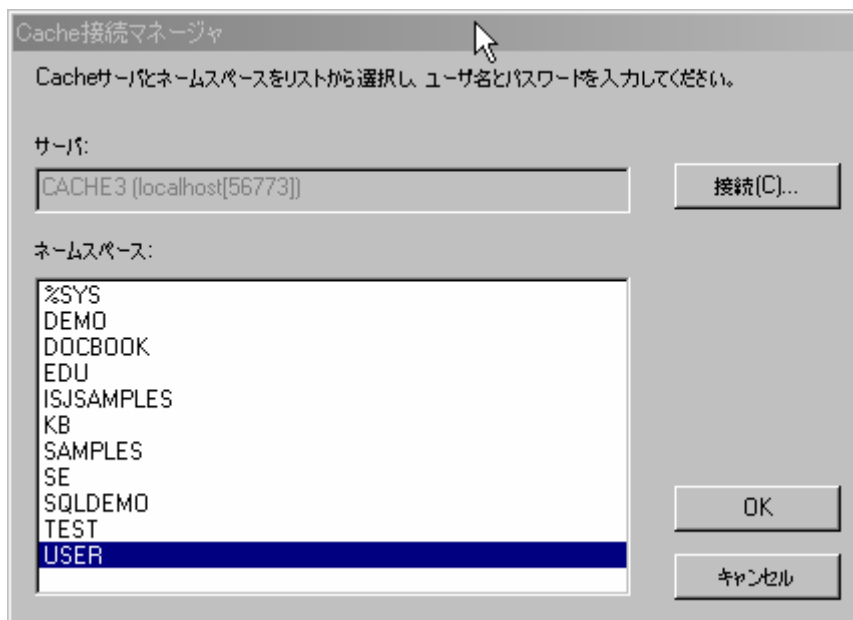
最後に、**PhoneNumber** は、操作 **RetrieveByContactName** を含みます。これは、**PhoneNumber** に追加して、メソッドと同じようにアクセスする定義済み SQL クエリです。

## Chapter4

## ネームスペースへの接続およびプロジェクトの作成

Cache スタジオ内ですべての開発作業を行います。スタジオを起動し、スタジオをネームスペースに接続し、新規プロジェクトを作成する手順を以下に示します。

1. Cache キューブをクリックして**[スタジオ]**を選択し、Cache スタジオを開きます。
2. 作業するネームスペースに切り替えます。**[ファイル]**→**[ネームスペース変更]**をクリックし、表示されるリストからネームスペースをクリックします。また、新規ネームスペースも作成できます。既定ではスタジオは最後に接続したネームスペースに接続します。初めてスタジオを起動する場合は、USER ネームスペースに接続します。これは作業に適した場所です。



3. ユーザ名とパスワードの要求を無視し、**[OK]**をクリックします。Cache スタジオが起動します。
4. **[ファイル]**→**[新規プロジェクト]**をクリックし、新規のプロジェクトを作成します。これにより Project1 という名前のプロジェクトが作成されますが、保存されません。**[ファイル]**→**[プロジェクト保存]**をクリックすることによりプロジェクトを保存します。このとき、Contacts などの記述的な名前をプロジェクトに付けることができます。

### Note:

Cache スタジオの使用に関する詳細は、Cacheドキュメント: [Cache ツールとユーティリティ] - [\[Cache スタジオの使用法\]](#)を参照してください。

## Chapter5

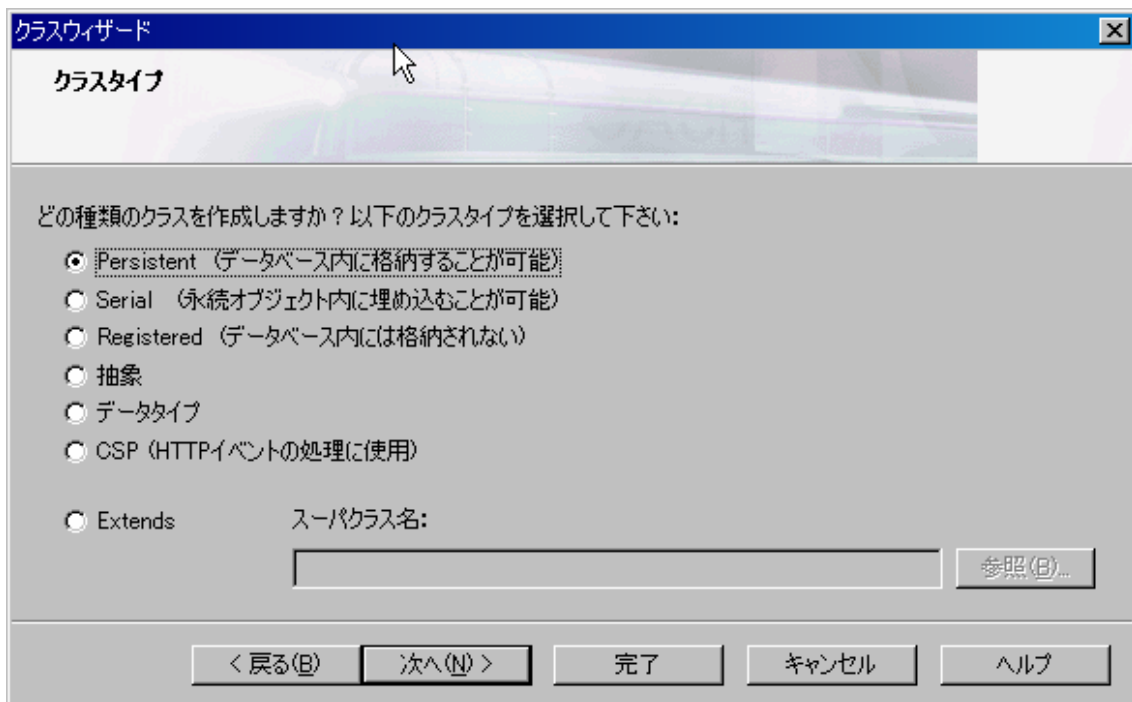
## 新規クラスの生成

最初にスタジオの**新規クラス・ウィザード**を使用して **Contact** クラスを作成します。

1. 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。
2. **[新規作成]** ダイアログの **[一般]タブ** をクリックします。
3. **[Cache クラス定義]** アイコンをクリックし、**[OK]** をクリックします。

ウィザードが起動した後、以下の手順を実行して **Contact** クラスを作成します。

1. ウィザードの最初の画面で、パッケージ名とクラス名を入力します。パッケージ名は、**ContactDB** にします。クラス名は、**Contact** である必要があります。
2. 次の画面で、クラス・タイプの **Persistent** をクリックしてから **[次へ]** をクリックします。これは不可欠です。 **Persistent** クラスのみがデータベース・テーブルを表現することができます。



3. 次の画面で、**[このクラスは自動データ入力をサポートする]** チェック・ボックスにチェックを付けます。
4. **[完了]** ボタンをクリックします。Caché クラス定義が作成されました。

## Chapter6

## クラスへのプロパティの追加

2つのプロパティを **Contact** に追加します。これらのプロパティは Name と ContactType です。**新規プロパティ・ウィザード**を使用してプロパティを追加するか、**クラス・エディタ**で直接コードを記述して作成できます。

**新規プロパティ・ウィザード**を使用する手順を以下に示します。

1. スタジオのメニュー・バーの **[クラス]→[追加]→[新規プロパティ]** をクリックして、ウィザードを起動します。
2. ウィザードの最初の画面で、プロパティ名 (Name および ContactType) と、必要に応じて、簡単な説明を入力します。説明はクラス定義にコメントとして表示されます。 **[次へ]** ボタンをクリックします。
3. ウィザードの 2 番目の画面でプロパティのタイプを入力します。

Name と ContactType の両方について、**[単一値タイプ]** をクリックし、**[参照]** をクリックし、[システム・データ型] の下の [%Library] フォルダを開き、**String** をクリックします。**[OK]** をクリックしてから **[次へ]** をクリックします。

4. ウィザードの 3 番目の画面では、プロパティのさまざまな特性を選択することができます。

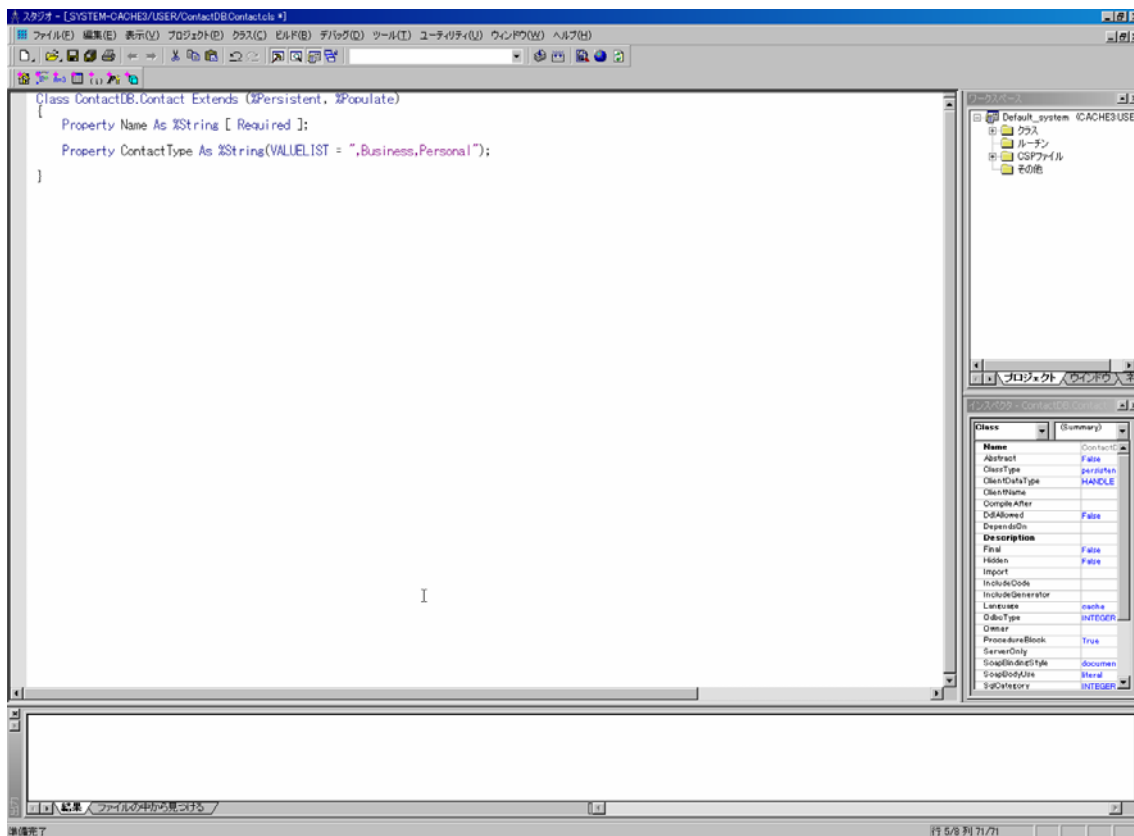
Name については、**[必須]** をクリックしてから **[次へ]** をクリックします。ContactType については、既定のままにします (ボックスを選択しません)。 **[次へ]** ボタンをクリックします。

5. ウィザードの 4 番目の画面では、プロパティのさまざまなパラメータを設定することができます。

Name については、既定のままにして、**[完了]** ボタンをクリックします。ContactType については、**VALUELIST** パラメータを設定します。このパラメータを設定すると、ContactType の可能な値がリストする値に制限されます。「**,Business,Personal**」と入力します。これにより ContactType の値は Business と Personal に制限されます。最初にコンマを付けることにより、コンマがリストの区切り文字であることをコンパイラに指示することに注意してください。このため、「Business」と「Personal」の間にはスペースではなく、コンマを入れます。 **[完了]** ボタンをクリックします。

6. プロパティのこれらの手順を完了するたびに、プロパティ宣言がスタジオの**クラス・エディタ**に表示されます。

完了すると、結果のクラス定義は以下のように表示されます。



**Note:**

スタジオの**クラス・エディタ**は id プロパティを表示しません。

## Chapter7

## 演習： PhoneNumber クラスの作成

演習として、アプリケーションの **PhoneNumber** クラスを作成します。クラスは、以下の要件を満たす点で **Contact** と似ています。

- クラスは、**Contact** と同じパッケージに所属します。
- その **[クラス・タイプ]** は Persistent です。
- 自動データ入力をサポートします。

さらに、**PhoneNumber** は以下の 2 つのプロパティを含みます。

プロパティ名	タイプ	パラメータ
PhoneNumberType	%String	VALUELIST=,Business,Home,Mobile,Fax
Number	%String	—

## Chapter8

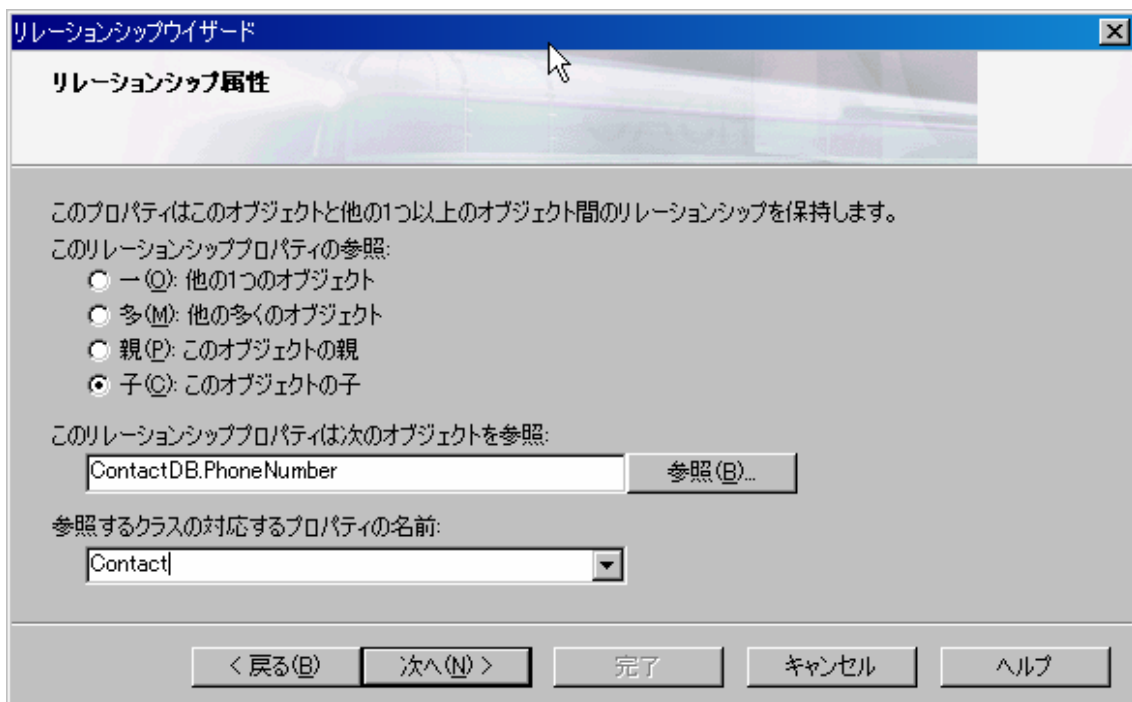
## リレーションシップの作成

**Contact** と **PhoneNumber** のリレーションシップを作成します。ここでも、**新規プロパティ・ウィザード**を使用するのが最も簡単です。必要に応じて、**クラス・エディタ**を使用して直接プロパティ定義を作成することもできます。

親子リレーションシップを作成する場合は、新規プロパティをそれぞれのリレーションシップ・クラスに追加します。親クラスには、子クラス・タイプのオブジェクトの配列を追加します。子クラスには、親クラス・オブジェクトを参照するプロパティを追加します。

リレーションシップを作成する手順を以下に示します。

1. スタジオの **Contact** をオープンし、**[クラス]→[追加]→[新規プロパティ]**を使用して**新規プロパティ・ウィザード**を起動します。
2. ウィザードの最初の画面で、プロパティに PhoneNumbers などの名前を割り当てます。**[次へ]**をクリックします。
3. **[プロパティ・タイプ]**画面で**[リレーションシップ]**をクリックします。**[次へ]**をクリックします。
4. **[リレーションシップ属性]**画面が以下のように表示されます。



この画面で、以下を指定します。

- このリレーションシップ・プロパティは、子：このオブジェクトの子を参照します。
  - このリレーションシップ・プロパティは、ContactDB.PhoneNumber のタイプのオブジェクトを参照します。
  - 参照されたクラス内の対応するプロパティの名前は Contact です。
  - **[次へ]**をクリックします。
5. **[追加変更]**画面で[ContactDB.PhoneNumber に新規プロパティ“Contact”を作成する]ボックスを選択します。 **[完了]** をクリックします。

ウィザードは以下の宣言を **Contact** に追加します。

```
Relationship PhoneNumbers As ContactDB.PhoneNumber
```

```
[ Cardinality = children, Inverse = Contact ];
```

ウィザードは以下の宣言を **PhoneNumber** に追加します。

```
Relationship Contact As ContactDB.Contact
```

```
[ Cardinality = parent, Inverse = PhoneNumbers ];
```

## Chapter9

## データベースへの入力の準備

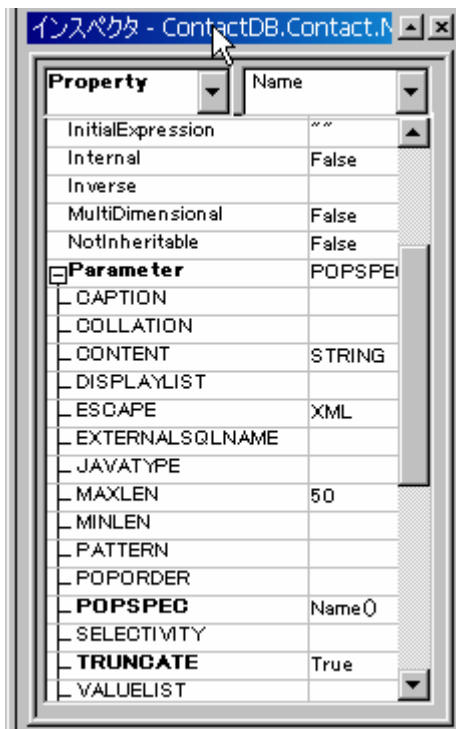
Cachéは、アプリケーションのテストに適したデータをデータベースに入力するためのユーティリティ・メソッドを提供します。具体的には、[%Library.Populate](#) と [%Library.PopulateUtils](#)には、一般に使用されるプロパティのための適切なフォームを持つランダム・データを生成するメソッドが含まれます。 [PhoneNumber](#)と[Contact](#)の両方は、[%Library.Populate](#)を拡張したもので、これらのプロパティに対してそれらの入力メソッドが使用可能です。

プロパティの POPSPEC パラメータを使用してそれを入力するためのメソッドを割り当てます。

[Contact](#) と [PhoneNumber](#) の両方に対して以下の手順を実行します。

1. スタジオでクラス([Contact](#)、[PhoneNumber](#))をオープンします。
2. **[表示]**→**[インスペクタ]**をクリックして、**インスペクタ**をオープンします。既定では、**インスペクタ**は**クラス・エディタ**の右側に表示されます。
3. **インスペクタ**の上部にある左側のドロップダウン・リストの**[プロパティ]**をクリックし、次に右側のドロップダウン・リストからプロパティ名 (Name、PhoneNumber)をクリックします。
4. **[パラメータ]**リストを展開します。
5. **POPSPEC** パラメータの横に適切なメソッド名を入力します。NameにName()を入力します。PhoneNumberにUSPhone()を入力します。
6. **クラス・エディタ**にカーソルを戻して、マウスをクリックします。これにより、**クラス・エディタ**は**インスペクタ**と同期が取られます。

Name の POPSPEC パラメータを設定する場合、**インスペクタ**は以下ようになります。



スタジオは、Name のプロパティ宣言を自動的に以下のように更新します。

```
Property Name As %String(POPSPEC = "Name() ") [ Required ];
```

**Note:**

Populate ユーティリティは、可能な値のリストからランダムに値を選択することにより、プロパティ ContactType および PhoneNumberType に VALUELIST パラメータの値の1つを自動的に生成します。このため、これらのプロパティに POPSPEC パラメータを設定する必要はありません。Caché Populate ユーティリティ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オブジェクトの使用法] - [\[Cachéデータ生成ユーティリティ\]](#) を参照してください。

## Chapter10

## コンパイルと生成

クラスをコンパイルするには、スタジオでクラスをオープンして、スタジオのメニュー・バーで[ビルド] → [コンパイル]をクリックします。 **Contact** と **PhoneNumber** のコンパイルが正常に完了したら、データを入力することができます。 Caché ターミナルからデータ生成ユーティリティを実行します。

**Contact** と **PhoneNumber** について、順番に以下の手順を実行します。

1. Caché キューブからターミナルを起動します。
2. ターミナル・プロンプトで、**zn** コマンドを使用して、クラスが格納されているネームスペースに切り替えます。例えば、SAMPLES でターミナルを開いており、クラスが USER にある場合は、次のように入力します。

```
SAMPLES>zn "USER"  
USER>
```

3. 次に、データ生成ユーティリティを使用して **Contact** と **PhoneNumber** のインスタンスを作成します。5つの **Contact** インスタンスと5つの **PhoneNumber** インスタンスを作成するコマンドは、以下のとおりです。

```
USER>do ##class(ContactDB.Contact).Populate(5,1)  
USER>do ##class(ContactDB.PhoneNumber).Populate(5,1)
```

このコマンドでは、PackageName.ClassName 構文を使用して、パッケージ名とクラス名を指定する必要があります。 **Populate** の最初の引数(この場合は 5)は、作成するインスタンスの数を示します。 **Populate** の2つ目の引数で 1 を渡すと、各インスタンスが作成されたときに、ユーティリティからフィードバックが提供されます。

次の画像は、USER ネームスペースに作成した、データ入力先となる **Contact** の5つのインスタンスを示しています。

```

Cache TRM:2664 (CACHE3)
ファイル(F) 編集(E) ヘルプ(H)

USER>do ##class(ContactDB.Contact).Populate(5,1)
Object saved...1
Object saved...2
Object saved...3
Object saved...4
Object saved...5
TABLE: ContactDB.Contact
  Current Extentsize = 100000
  Calculated Extentsize = 5                               Updated
  FIELD: ContactType
    Current Selectivity = <Not Specified>
    Calculated Selectivity = 33.3333%
    Table definition updated.
    Class definition updated.
  FIELD: Name
    Current Selectivity = <Not Specified>
    Calculated Selectivity = 20.0000%
    Table definition updated.
    Class definition updated.
USER>

```

後で **Contact** または **PhoneNumber** のすべてのインスタンスを削除する場合は、**%KillExtent** を使用します。次のコマンドでは、USER から **ContactDB.Contact** のすべてのインスタンスが削除されます。

```
User>do ##class(ContactDB.Contact).%KillExtent()
```

**Note:**

**Contact** のインスタンスを作成する前に、**PhoneNumber** のインスタンスを作成することはできません。これは、**Contact** を親とする親子リレーションシップが2つのクラス間に確立されているためです。各 **PhoneNumber** インスタンスは、**Contact** インスタンスに属している必要があります。

## Chapter11

## データベース・テーブルの表示

Cachéシステム管理ポータルを使用して、生成した **Contact** および **PhoneNumber** テーブルを表示することができます。手順は以下のようになります。

1. Caché キューブからシステム管理ポータルを起動します。
2. **[データ管理]**列で、**[SQL]**をクリックします。
3. **[ネームスペース]**列で、**[ContactDB]**をクリックするか、**Contact** および **PhoneNumber** を含むネームスペースをクリックします。
4. **[SQL 操作]**列で、**[SQL スキーマを参照]**をクリックします。
5. スキーマ名 **ContactDB**、またはアプリケーションに指定したネームスペース名の行にある**[テーブル]**をクリックします。
6. **[Contact]**行で**[テーブルを開く]**をクリックして **Contact** データを表示するか、**[PhoneNumber]**行で**[テーブルを開く]**をクリックして **PhoneNumber** データを表示します。

#	ID	ContactType	Name
1	1	Business	Larson, Geoffrey M.
2	2	Personal	Edison, Alice D.
3	3	Personal	Chadwick, Emily D.
4	4	Personal	Gold, Zoe T.
5	5	Personal	Jung, Patricia I.

テーブルの ID 列に注意してください。この列には **Contact** インスタンスのオブジェクト ID が含まれます。

**Note:**

パッケージ名が User である場合、スキーマ名は SQLUser です。

## Chapter12

## データベースの問い合わせ

Caché システム管理ポータルでは、データベース・テーブルに対して直接 SQL クエリを実行することができます。手順は以下のようになります。

1. Caché キューブからシステム管理ポータルを起動します。
2. **[データ管理]**列で、**[SQL]**をクリックします。
3. **[ネームスペース]**列で、**[ContactDB]**をクリックするか、**Contact** および **PhoneNumber** を含むネームスペースをクリックします。
4. **[SQL 操作]**列で、**[SQL 文の実行]**をクリックします。
5. **[SQL クエリ]**テキスト・ボックスに SQL クエリを入力します。
6. **[クエリ実行]**ボタンをクリックします。クエリ結果がページの下部に表示されます。

以下のフォームでネームスペース USER で実行するSQLクエリを入力してください:

SQLクエリ: `SELECT * FROM ContactDB.Contact WHERE ContactType = 'personal'`

データ表示モード: 論理モード

最大行数: 1000

クエリ実行    クエリプラン表示    クエリ履歴    クエリビルダ

実行されたSQLクエリの結果を以下に表示します: 最終更新: 2007-05-23 20:02:02.364

SQLCODE: 100 列数: 4 パフォーマンス: 0.001 秒 29 グローバル参照

#	ID	ContactType	Name
1	2	Personal	Edison,Alice D.
2	3	Personal	Chadwick,Emily D.
3	4	Personal	Gold,Zoe T.
4	5	Personal	Jung,Patricia I.

差了

ページが表示されました


## Chapter13

## クラス・クエリの追加

次に、クラス・クエリを **PhoneNumber** に追加します。クラス・クエリは、一連のプロパティ値を取得するメカニズムをアプリケーションに提供します。この場合、クエリは Name プロパティに特定の値を持つ **Contact** インスタンスに関連付けられたすべての **PhoneNumber** インスタンスのオブジェクト ID を返します。クエリを追加した後、**PhoneNumber** を再コンパイルします。

**新規クエリ・ウィザード**を使用するか、**クラス・エディタ**で直接コードを記述してクエリを追加できます。

**新規クエリ・ウィザード**を使用する手順を以下に示します。

1. スタジオのメニュー・バーの **[クラス]→[追加]→[新規クエリ]** をクリックして、ウィザードを起動します。
2. ウィザードの最初の画面で、クエリ名 **RetrieveByContactName** を割り当てます。次に、クエリが SQL 文に基づくことを指定します。 **[次へ]** をクリックします。
3. ウィザードの 2 番目の画面で入力パラメータを指定します。これを実行するには、画面の右側にある 4 つのボタンの最上部のボタンをクリックします (  )。パラメータに name などの名前を割り当てます。プロパティのタイプは、 **%String** である必要があります。 **[OK]** をクリックしてから **[次へ]** をクリックします。
4. ウィザードの 3 番目の画面でクエリによって返されたプロパティ (列) を選択します。 **%ID** をダブルクリックします。 **[次へ]** をクリックします。
5. ウィザードの最後の画面は、クエリ条件を作成するのに役立ちます。ウィザードを使用せずに、手動でクエリ条件を追加します。 **[完了]** ボタンをクリックします。

ウィザードは以下の宣言を **PhoneNumber** に追加します。

```
Query RetrieveByContactName(name As %String) As %SQLQuery (CONTAINID = 1)
{
SELECT %ID FROM PhoneNumber
}
```

クエリを完了するには、クエリ条件を追加する必要があります。つまり、SQL WHERE 節を追加する必要があります。order by 節を追加して結果をグループ化することもできます。ここでは、2 つの節を考えます。

```
Where (Contact->Name)=:name
order by PhoneNumberType
```

この節は、Name プロパティの値がクエリに渡された引数と一致する **Contact** テーブルの行を参照する **PhoneNumber** テーブルの行を選択します。

完了すると、結果のクエリは以下のように表示されます。

```
Query RetrieveByContactName(name As %String) As %SQLQuery (CONTAINID = 1)
{
SELECT %ID FROM PhoneNumber
Where (Contact->Name)=:name
order by PhoneNumberType
}
```

**Note:**

クエリのWHERE節は、Cachéの"—>"演算子を使用します。この演算子によって JOIN構文がわかりやすくなります。"—>"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] - [特別な機能] - [\[暗黙結合\]](#) を参照してください。

クラス・クエリ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。

## Chapter14

## 要約とプレビュー

チュートリアルでは、ここまでで以下のことを行いました。

- Contact Management システムのデータ・モデル、**Contact** と **PhoneNumber** を作成しました。
- Caché の統合データ・アーキテクチャにより、どのようにデータ・モデルを一連の関連するクラスとして(オブジェクト指向の視点)、および一連の関連テーブルとして(リレーショナルの視点)扱うことができるかを学習しました。
- Contact Management システムにサンプル・データを入力しました。

次に、Contact Management システムの Web ベースのインターフェースを作成します。 Caché Server Page (CSP)が、動的な Web アプリケーションを開発するために、どのようにオブジェクト指向プログラミングのパワーとスピードを拡張するかを理解することができます。

ContactPage.csp と PhoneNumberPage.csp の 2 つの CSP ページを作成します。

これらの CSP ページによりアプリケーションのユーザは以下のことができます。

- 名前、タイプ、および関連する電話番号のすべてなどの連絡先の情報を表示し、編集します。
- アプリケーションに新規連絡先を追加します。
- 連絡先に新しい電話番号を追加します。
- 検索ページを起動して、ビジネスまたは個人のいずれかのタイプによって連絡先のすべてのインスタンスを並べ替えます。

## Chapter15

## CSP とは

Caché Server Page (CSP)は、動的データ駆動型の Web アプリケーションを迅速に開発するためのテクノロジーです。個別のページはHTTP要求を処理し、HTMLを生成することができる Caché クラスです。開発者は標準 HTML ファイルを使用して CSP を作成することができます。CSP コンパイラはこれらの HTML ファイルを Caché クラスに変換します。CSP は、HTML/CSP ファイルに追加することができる複数の特別な要素も提供します。CSP コンパイラはこれらの要素を Caché コードに変換します。Caché が HTML ページを生成するとき、要素は動的な振る舞いを追加します。以下のテーブルは Contact Management システムに対するインタフェースを作成するために使用する CSP 要素を説明します。

要素	説明
<code>#{Expr}#</code>	CSP 式。Expr は Caché Basic または ObjectScript で記述された式です。Caché は、式を評価し、その結果を生成する HTML ページに挿入します。
<code>&lt;script language=("CACHE" "BASIC") runat="Server"&gt; ObjectScript または Basic Code &lt;/script&gt;</code>	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 はコードを実行し、その結果を生成する HTML ページに挿入します。コードは、Caché Basic または ObjectScript のどちらでもかまいません。
<code>&lt;script language=("CACHE" "BASIC") method="" arguments="" returntype=""&gt; ObjectScript または Basic Code &lt;/script&gt;</code>	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。
<code>&lt;CSP: XXX&gt;</code>	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 CSP タグ。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。

**Note:**

対応する Caché クラスを直接コーディングすることによっても、CSP を開発できます。CSP の基礎や上のテーブルに示した要素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 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。

## Caché Server Page の作成と結合されたフォームの追加

以下の手順を実行して ContactPage.csp を作成します。

1. スタジオのメニュー・バーの **[ファイル]** → **[新規作成]** をクリックして、**新規クラス・ウィザード** を起動します。
2. **[新規作成]** ダイアログ・ボックスの **[CSP ファイル]** タブをクリックします。
3. **[Caché Server Page]** アイコンをクリックし **[OK]** をクリックします。
4. ウィザードによって作成されたスケルトン HTML ページを調査します。
5. ファイルを CSP¥User ディレクトリに ContactPage.csp として保存します。

次に、**Web フォーム・ウィザード** を使用して ContactPage.csp にフォームを追加します。フォームにはコンタクト情報が表示され、更新することができます。ウィザードは、フォームを **Contact** のインスタンスに結合するページにコードを追加します。結合メカニズムには、フォームのフィールドに **Contact** からのデータを入力し、基本的なデータの妥当性検証を提供し、フォーム・データをサーバに保存するコードが含まれます。

以下の手順を実行して ContactPage.csp にフォームを追加します。

1. スタジオの **クラス・エディタ** で、ContactPage.csp の `<BODY></BODY>` タグ内にカーソルを置きます。
2. **[挿入]** → **[フォーム・ウィザード]** をクリックして、**Web フォーム・ウィザード** を起動します。
3. 最初の画面で **[次へ]** をクリックします。
4. 次の画面で、フォームを **Contact** に結合します。 **[次へ]** ボタンをクリックします。
5. 次の画面で、含めるプロパティをクリックします。プロパティは、ContactType、Name、および %Id() です。 **[次へ]** ボタンをクリックします。
6. 次の画面で、それぞれのプロパティのラベルと読み取り専用設定を指定することができます。既定の設定のままにして、 **[次へ]** ボタンをクリックします。
7. 最後の画面で **[完了]** をクリックします。
8. ウィザードが ContactPage.csp に追加したコードを調査します。

## Chapter17

## フォーム・ウィザードのコード: &lt;CSP:Object&gt;および&lt;FORM&gt;の調査

**Web フォーム・ウィザード**は、フォームとその要素を含むテーブルを定義する HTML だけでなく、複数の CSP 要素を追加します。

最初に、<CSP:OBJECT>タグに注意してください。このタグは、オブジェクト(この場合は CSP ページに使用するための **Contact** インスタンス)をオープンするコードを生成します。

```
<csp:object name="objForm" classname="ContactDB.Contact"
           OBJID=#(%request.Get("OBJID"))#>
```

<CSP:OBJECT>タグは以下の複数の重要な属性をサポートします。

- name 属性は名前(この場合は objForm)をオブジェクトに割り当てます。名前は CSP ページでオブジェクトを参照するために使用することができます。
- classname 属性は、オブジェクトのクラスを識別します。クラスは、PackageName.ClassName 形式を使用して識別します。
- OBJID属性は、ページ上で開くオブジェクトの特定のインスタンスのオブジェクトIDを識別します。この例では、#( )#を使用して%requestオブジェクトからOBJID値を取得します。CSP は%[CSP.Request](#)を使用して状態を維持します。つまり、ページ要求間でデータを共有します。

HTML <FORM>タグにも注意してください。

```
<form name="form" cspbind="objForm" cspjs="All"
      onsubmit='return form_validate();'>
```

CSP は以下のものを含む<FORM>タグの追加属性を提供します。

- <CSP:OBJECT>タグによってオープンしたオブジェクトにフォームを結合する cspbind 属性。
- HTML ページにさまざまな JavaScript 関数を含めるよう Caché に指示する cspjs 属性。
- フォームが送信されたときに呼び出す Caché 付属 JavaScript 関数を識別する onsubmit 属性。

**Note:**

状態および[%CSP.Request](#)の維持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 での HTTP 要求] - [\[%CSP.Requestオブジェクト\]](#)を参照してください。 cspjsにより使用可能なJavaScript関数についてさらに学習するには、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] - [HTMLタグ] - [\[<FORM>\]](#) を参照してください。

## Chapter18

## フォーム・ウィザードのコード、&lt;CSP: Search&gt;および&lt;input&gt;の変更

ウィザードのコードに以下の変更を行います。 <CSP: Search>タグから開始します。

```
<csp:search name="form_search" classname="ContactDB.Contact" where="%Id()"
  options="popup,nopredicates" onselect="update">
```

このタグはウィザードが ContactPage.csp に生成する **[検索]** ボタンによって起動できる検索ページを作成します。検索ページでは、ユーザはデータベースに格納された **Contact** インスタンスを検索することができます。既定では、ページはオブジェクト ID を使用して検索します。

ContactType を使用して検索するページをカスタマイズするには、タグが以下のようになるように <CSP: SEARCH> の属性の値を変更します。

```
<csp:search name="form_search" classname="ContactDB.Contact"
  where="ContactType" select="ContactType,Name" predicates="select">
```

このウィザードでは、このウィザードで作成された JavaScript 関数を呼び出す一連のボタンも生成します。これらの関数は、フォームを保存し、フォームをクリアし、検索ページを起動します。

```
<input type="button" name="btnClear" value="クリア" onclick='form_new();'>
<input type="button" name="btnSave" value="保存" onclick='form_save();'>
<input type="button" name="btnSearch" value="検索" onclick='form_search();'>
```

以下の JavaScript イベント・ハンドラ・コードを **[保存]** および **[クリア]** ボタンの両方に追加します。このコードは、ユーザが新しい **Contact** を保存するか、フォームをクリアするときに強制的にページ全体を更新します。以下は、変更されたボタンのコードです。

```
<input type="button" name="btnClear" value="クリア"
  onclick="if (form_new()==1){self.document.location='ContactPage.csp?'};">
<input type="button" name="btnSave" value="保存"
  onclick=
    "if (form_save()==1)
      {self.document.location='ContactPage.csp?OBJID='+form.sys_Id.value};">
```

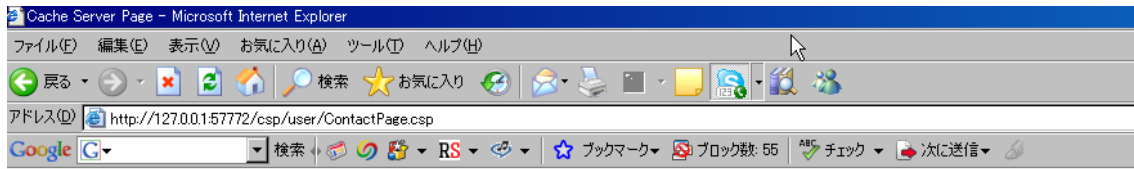
**Note:**

<CSP: Search>についてさらに学習するには、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] - [\[<CSP: SEARCH>\]](#) を参照してください。

## Chapter19

## ContactPage.csp および検索ページ

ContactPage.csp をコンパイルするには、スタジオの[エディタ]メニュー・バーで **[ビルド]→[コンパイル]** をクリックします。スタジオのメニュー・バーで **[表示]→[ウェブページ]** をクリックして、Web ブラウザで ContactPage.csp を開きます。ContactPage.csp は以下ようになります。



## ContactDB.Contact

ContactType:

\*Name:

ID:

(\*は必須フィールドを示します)

**[検索]** ボタンをクリックすると、検索ページが起動されます。 **[Personal]** をクリックし、 **[検索]** をクリックすると、ContactType が Personal であるデータベース内のすべての連絡先の名前がリストされます。

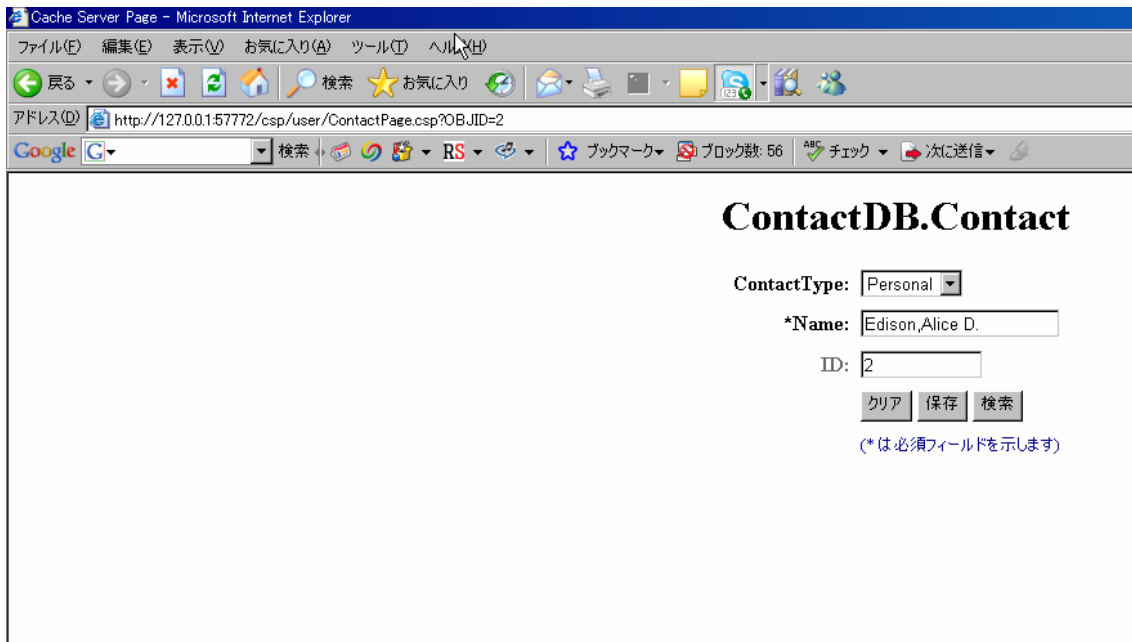
## Caché 検索

以下の条件を満たす、**ContactDB.Contact** の全てのインスタンスを検索します:

ContactType

ContactType	Name
<a href="#">Personal</a>	Edison,Alice D.
<a href="#">Personal</a>	Chadwick,Emily D.
<a href="#">Personal</a>	Gold,Zoe T.
<a href="#">Personal</a>	Jung,Patricia I.

ContactPage.csp のフォームにインスタンスをロードするリンクのクリック。フォームにロードすると、**Contact** インスタンスのプロパティ値を更新できるようになります。



## Chapter20

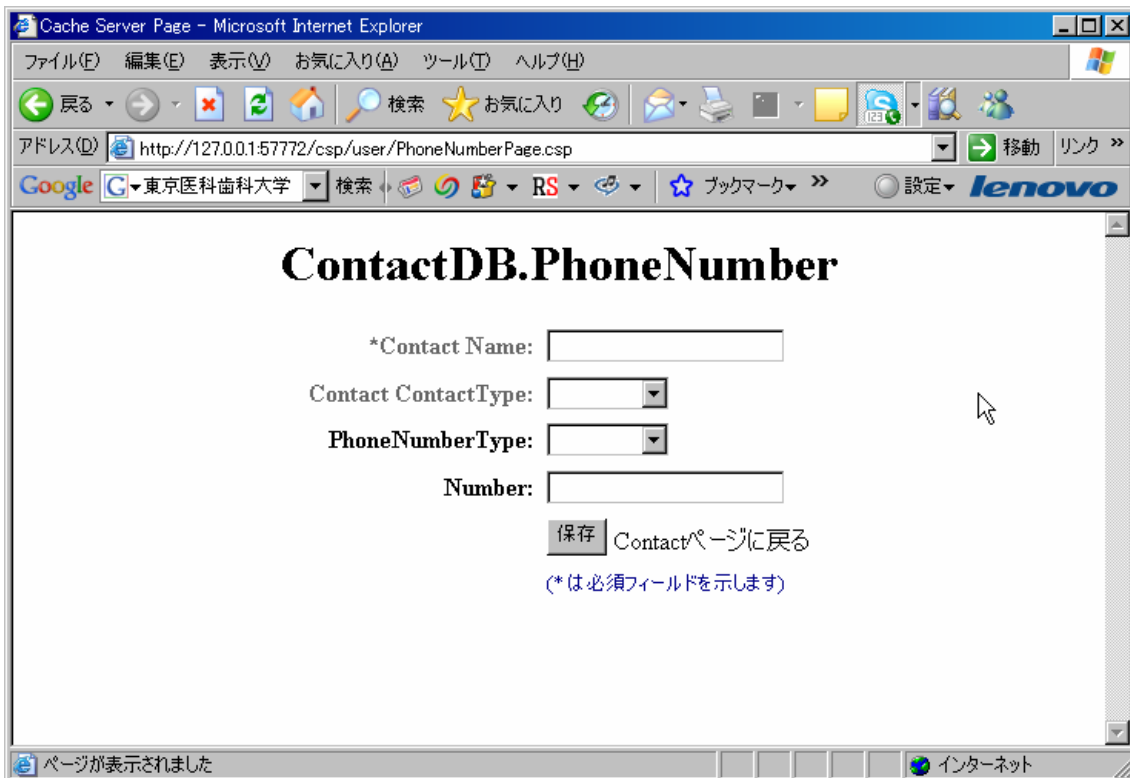
## 演習： PhoneNumberPage.csp の作成

演習として、PhoneNumberPage.csp という名前の CSP ページを作成します。 ページは以下の条件を満たす必要があります。

- **PhoneNumber** に結合された HTML フォームを含む。
- Number および PhoneNumberType プロパティは、編集可能である。
- **PhoneNumber** と関連する **Contact** の Name と ContactType を表示する。
- Name および ContactType プロパティは、読み取り専用である。
- 「Contact ページに戻る」という 1 行のテキストが表示される。 このテキストは、このページを ContactPage.csp に接続するハイパーリンクに使用します。

**Web フォーム・ウィザード**は、<csp: search>タグと共に**[検索]**および**[クリア]**ボタンをページに自動的に追加します。 アプリケーションはこの機能を使用しないので、ボタンとタグを削除する必要があります。 対応する<input type="button">タグを削除することによりボタンを削除します。 <csp: search>タグも削除します。

完成した PhoneNumberPage.csp ページは、Web ブラウザで表示した場合以下のようになります。



## Chapter21

## 演習： &lt;CSP:Query&gt; および &lt;CSP:While&gt; の使用

次に、連絡先に格納された電話番号を表示する ContactPage.csp にテーブルを追加します。これを行うには、<CSP:Query>、<CSP:Object> および <CSP:While> の 3 つの CSP タグを使用します。

このスケルトン・コードはこれらの 3 つのタグを組み合わせて **Contact** のすべての **PhoneNumber** インスタンスを繰り返します。それぞれを開いて情報を取得します。

```
<csp:query name="query" classname="ContactDB.PhoneNumber"
           queryname="RetrieveByContactName" P1="#(objForm.Name)#">
  <!--Additional Code Here-->
<csp:while counter=queryRow condition=query.Next()>
  <csp:object name="PhoneNumber" classname="ContactDB.PhoneNumber"
             OBJID=(query.Get("ID"))#/>
  <!--Additional code to be repeated here-->
</csp:while>
```

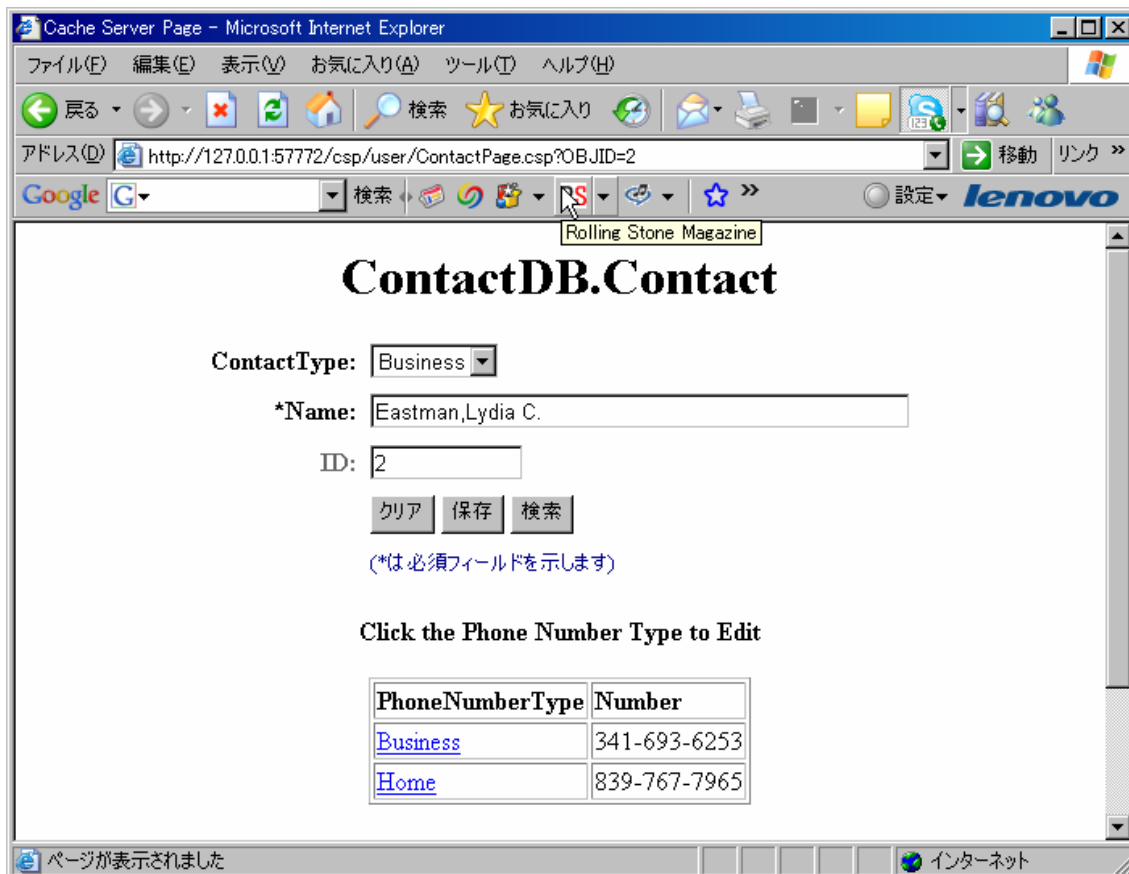
これから以下の点がわかります。

- <CSP:Query> は **PhoneNumber** に定義された **RetrieveByContactName** クエリを実行します。このクエリは指定された Name を持つ **Contact** オブジェクトに格納されたそれぞれの **PhoneNumber** オブジェクトのオブジェクト ID を返します。タグの name 属性はクエリによって返された結果セットの名前を提供します。P1 属性はクエリに渡されるパラメータを識別します。この場合、objForm によって参照される **Contact** オブジェクトの Name プロパティを渡します。
- <CSP:While> は while ループを作成します。タグの condition 属性は、[%Library.ResultSet Next](#) メソッドを使用してループの反復を制御します。ループは結果セットの最後の行に達するまで反復されます。
- <CSP:Object> は **PhoneNumber** インスタンスをオープンします。タグは、[%Library.ResultSet Get](#) メソッドを使用して現在の結果セット行の "ID" 列から object id 値を取得します。

演習として、上記のコードを ContactPage.csp に追加して、以下の条件を満たすように完成させます。

- それぞれの **PhoneNumber** インスタンスに対して、ページに PhoneNumberType および Number プロパティ値が表示されます。ヒント: プロパティ値にアクセスするには、# ( ) #を使用します。
- データはHTMLテーブルに表示されます。Cachéドキュメント: [\[Caché開発リファレンス\]](#) - [\[CSP HTML タグ・リファレンス\]](#) は、多数の重要なHTMLタグの使用法について説明しています。

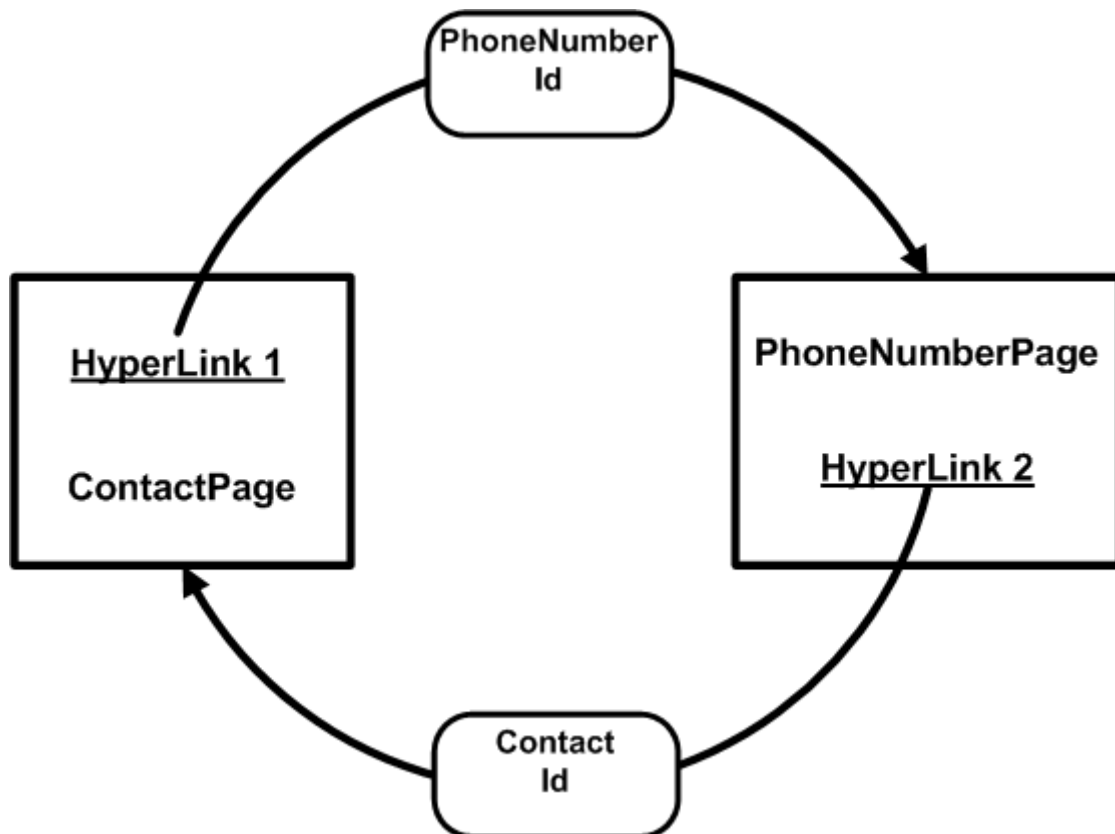
完了すると、ページは以下のように表示されます。



## PhoneNumberPage.csp への ContactPage.csp のリンク

このとき、Contact Management システムへのインターフェースは 2 つの Web ページから構成されます。アプリケーションを使用可能にするには、これらのページに接続するハイパーリンクを提供する必要があります。

ContactPage.csp を PhoneNumberPage.csp に接続するハイパーリンクは、PhoneNumberPage.csp が編集のためにオープンする **PhoneNumber** インスタンスのオブジェクト ID を提供する必要があります。同様に、PhoneNumberPage.csp を ContactPage.csp に接続するハイパーリンクは、**Contact** インスタンスのオブジェクト ID を備えた ContactPage.csp を提供する必要があります。 **%request** を使用して ID を新規ページに転送します。



以下のアンカー・タグ・コード ContactPage.csp を追加して PhoneNumberPage.csp へのリンクを作成します。ContactPage.csp の電話番号のテーブルに **PhoneNumber** インスタンスの PhoneNumberType プロパティを表示するコードの周囲にタグを配置します。

```
<a href=' PhoneNumberPage.csp?OBJID=#(PhoneNumber.%Id())#' > </a>
```

このコードについて、以下のことに注意してください。

- アンカー・タグ・コードは、URL クエリ・パラメータ `?Name=Value` を使用して **PhoneNumber** インスタンスのオブジェクト ID を **%request** オブジェクトに追加します。Name は OBJID で、Value はオブジェクト ID です。
- URL クエリ・パラメータ・コードは、`<CSP:Object>` タグおよび `#( )#` によって設定される **PhoneNumber** 参照を使用して **PhoneNumber** インスタンスのオブジェクト ID の値を取得します。
- `PhoneNumberPage.csp` の `<CSP:Object>` タグは、正しい **PhoneNumber** インスタンスをオープンするためにページが **%request** オブジェクトをロードし、使用するとき、**%request** オブジェクトからオブジェクト ID を取得します。

**Note:**

URLクエリ・パラメータ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 での HTTP 要求] - [\[%CSP.Requestオブジェクト\]](#) を参照してください。

**%request**ドキュメントについては、[%CSP.Request](#)を参照してください。

## Chapter23

## 演習: ContactPage.csp への PhoneNumberPage.csp のリンク

演習として、ContactPage.csp に接続するハイパーリンクを PhoneNumberPage.csp に追加します。ハイパーリンクは以下の条件を満たす必要があります。

- PhoneNumberPage.csp を ContactPage.csp にリンクする。
- そのアンカー・タグが、「クリックして Contact ページに戻ります」というテキストをラップする。
- URL クエリ・パラメータを使用して現在の **Contact** インスタンスのオブジェクト ID を ContactPage.csp に転送する。ヒント: 正しい **Contact** インスタンスの id を取得するには、以下のコードを使用します。

```
OBJID=#(objForm. Contact. %Id())#
```

## Chapter24

## フォーム処理パート 1

Contact Management システムには、現在新規電話番号を連絡先に追加するための機能がありません。この機能を 2 つのフェーズに分けて追加します。最初に、電話番号にデータを入力することができる ContactPage.csp にフォームを追加します。次に、フォームを処理し、データを保存する一対のスクリプトを追加します。

以下のフォームを ContactPage.csp に追加します。HTML テーブルを使用してフォーム要素をページ上にレイアウトします。

```
<form name="NewPhoneForm" method="POST" >
  <select name="PhoneNumberType" >
    <option value="Home">Home</option>
    <option value="Business">Business</option>
    <option value="Mobile">Mobile</option>
    <option value="Fax">Fax</option>
  </select>
  <input type="text" Name="Number" value="" />
  <input type="submit" Name="Submit" value="Save Number" />
</form>
```

Caché がページを生成するたびに、以下のフォーム処理スクリプトが実行されます。フォームが送信され、Submit の **%request** に格納された値がある場合、**AddPhoneNumber** メソッドが呼び出されます。このメソッドは、追加する 2 番目のフォーム処理スクリプトに定義されます。ページの **<HEAD>** と **<BODY>** セクション間の ContactPage.csp に以下のコードを追加します。

```
<script language="cache" runat="server">
if %request. Get("Submit") '="" {do ..AddPhoneNumber ()}
</script>
```

**Note:**

このスクリプトは Caché ObjectScript を使用します。

このスクリプトは **..AddPhoneNumber** を使用して **AddPhoneNumber** を呼び出します。 **..MethodName** 構文には、同じクラスに定義されたメソッドを呼び出すための ObjectScript 省略表現があります。この構文は、Basic の **Me.MethodName** および Java の **this.MethodName** 構文と同じです。

## フォーム処理パート 2

2 番目のスクリプトである CSP メソッドは、Caché Basic を使用して **AddPhoneNumber** を定義します。最初のフォーム処理スクリプトはこのメソッドを呼び出します。このメソッドは、ページで現在開いている **Contact** オブジェクトに関する情報および **%request** からの新規 **PhoneNumber** に関するフォーム情報を取得します。新規 **PhoneNumber** インスタンスを作成し、**Contact** に追加します。

以下の CSP メソッドを ContactPage.csp に追加します。

```
<script language=BASIC Method="AddPhoneNumber" arguments="">
Set phoneNumber= New ContactDB.PhoneNumber ()
Set contact = OpenId ContactDB.Contact(%request.Get("OBJID"))
If Not(contact="") Then
phoneNumber.PhoneNumberType=%request.Get("PhoneNumberType")
phoneNumber.Number=%request.Get("Number")
contact.PhoneNumbers.Insert(phoneNumber)
contact.%Save ()
End If
</script>
```

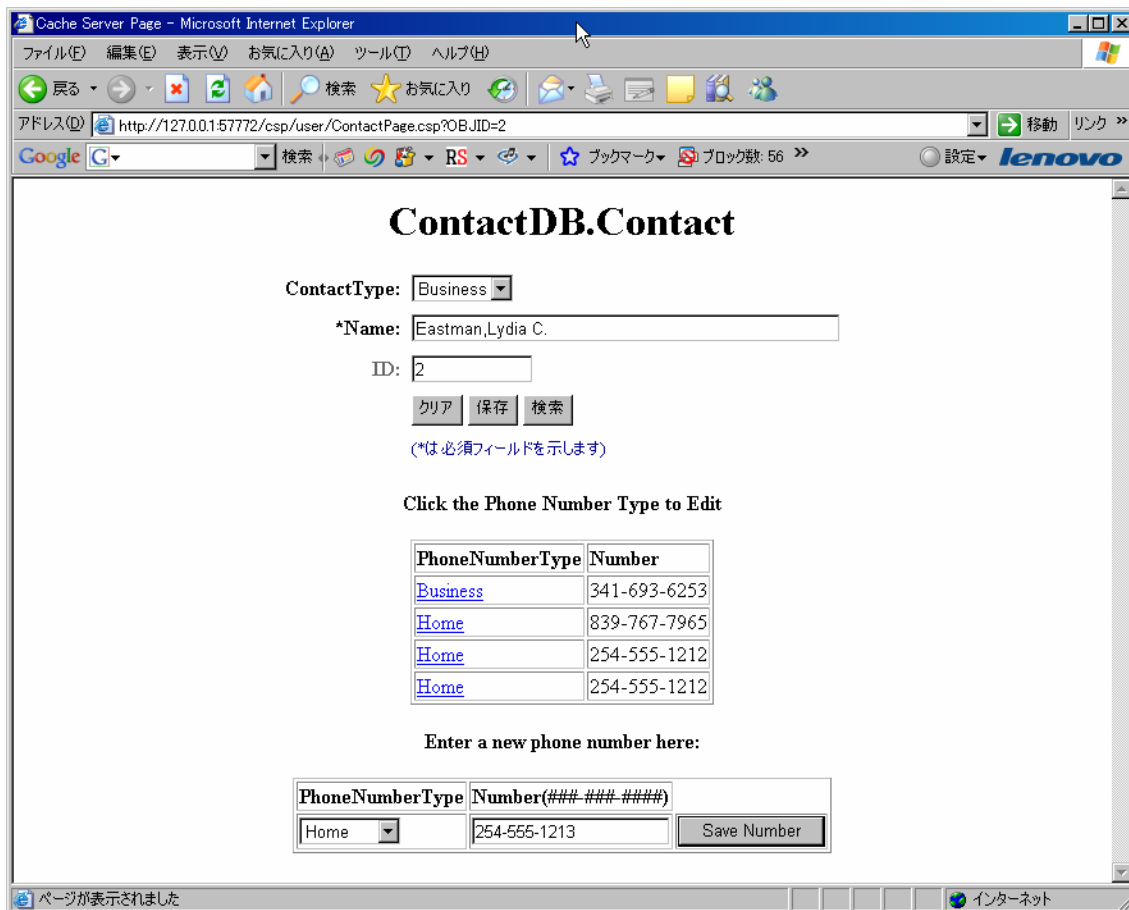
### Note:

どちらのフォーム処理スクリプトも、同じ言語、つまり Caché Basic または ObjectScript でコーディングすることができます。2 つのスクリプトは 1 つのスクリプトにまとめることができます。アプリケーションでは、分かりやすく説明するために、2 つのスクリプトと 2 つの言語を使用しています。CSP ページでの Caché Basic と ObjectScript の使用法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] - [\[CSP マークアップ言語\]](#) を参照してください。

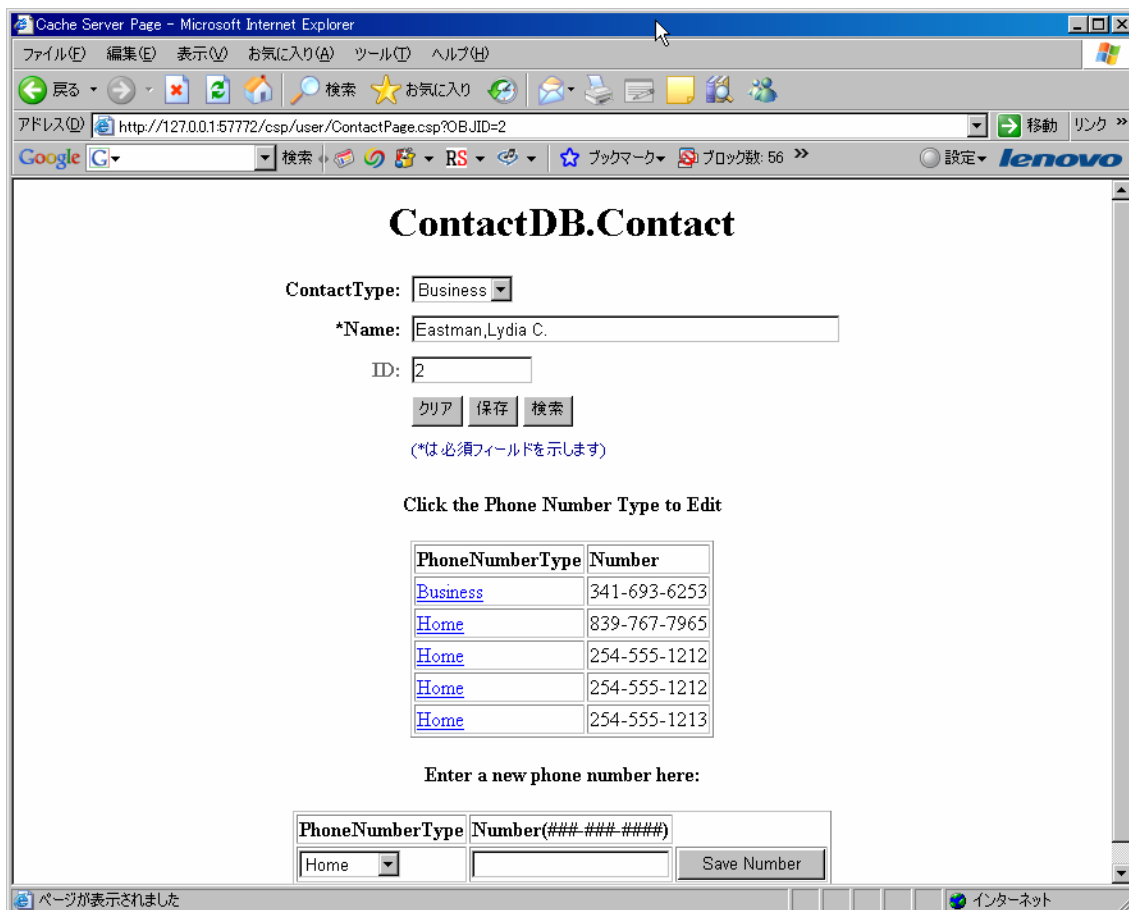
## Chapter26

## 完成した連絡先ページ

完了すると、ContactPage.csp は以下のように表示されます。



電話番号を追加し、**[Save Number]**をクリックすると、ページが再表示され、新規電話番号が電話番号リストに含まれます。



## 要約

このチュートリアルでは、Caché、CSP および Caché スタジオ開発環境を使用して、堅牢な Web 対応データベース・アプリケーションを迅速かつ簡単に作成する方法について学習しました。特に以下のことを学習しました。

- Caché 統一データ・アーキテクチャでは、データをオブジェクトとリレーショナル・テーブルの両方として扱うことができます。特定の作業に便利な方を選択することができます。
- Caché はアプリケーション開発のすべての分野をカバーする豊富なツールを提供します。
- Caché は、豊富で柔軟性の高いクラスとユーティリティを用意し、アプリケーション開発を迅速化し、自動データ生成などの機能を提供します。
- Caché Server Page(CSP)は、アプリケーション・データのための動的な Web ベースのインタフェースを開発するための迅速で効率的な手段を提供します。

## Appendix A

## サンプル・アプリケーション

ファイル QKSTutorial.xml には、作成済みの Contact Management アプリケーションが含まれます。このファイルは <cachsys>%dev%tutorials にインストールされます。

アプリケーションを USER ネームスペースにロードしたり、他のネームスペースにロードする場合は、以下の作業を行います。

1. Caché **スタジオ** を起動します。
2. **[ファイル]** メニューの **[ネームスペース変更]** をクリックして、アプリケーションをロードするネームスペースに接続します。
3. **[ツール]** メニューの **[ローカルからインポート]** をクリックし、**[XML]** を選択します。  
<cachsys>%dev%tutorials を検索します。
4. QKSTutorial.xml をクリックし **[開く]** をクリックします。

**Note:**

Windows で Caché を標準インストールしている場合、<cachsys> は C:%InterSystems%Cachsys になります。Unix または Linux の環境では、<cachsys> は /user/cachsys になります。