



Caché ファーストステップ・ガイド

(Caché Version 2015.1 ベース)

V1.0

2015 年 4 月

インターシステムズジャパン株式会社

目次

1.	はじめに	4
2.	ユーティリティ	6
2.1.	ドキュメント	7
2.2.	スタジオ	7
2.3.	ターミナル	8
2.4.	管理ポータル	8
3.	データベースとネームスペース	14
4.	最初のCachéプログラム	18
4.1.	Hello world	18
4.2.	パラメータを渡す	20
5.	Cachéオブジェクト	21
5.1.	クラスを作成	21
5.2.	プロパティの定義	25
5.3.	コンパイル	28
5.4.	インスタンスの生成・保存	28
5.5.	インスタンスの読み込み	29
5.6.	メソッドの定義	30
5.7.	計算プロパティ	31
5.8.	埋め込みオブジェクト	33
5.9.	Populate機能によるテストデータの生成	36
6.	Caché SQL	38
6.1.	SQLメニュー	39
6.2.	SQL計算フィールド	40
6.3.	MS-Accessからのアクセス(ODBC)	42
6.4.	クラスクエリ	44
6.5.	インデックスの生成	51
7.	最後に	52

図表目次

図 1 Cachéランチャーのメニュー	6
図 2 管理ポータル:システム管理	9
図 3 管理ポータル:システムエクスプローラー	10
図 4 管理ポータル:システムエクスプローラ:SQL	11
図 6 管理ポータル:クエリプランの表示	12
図 7 管理ポータル:システムオペレーション	13
図 8 管理ポータル:構成→システム構成	14
図 8 ネームスペース一覧	15
図 9 ネームスペースのデフォルトデータベースの確認	15
図 11 ネームスペースとデータベース 定義例	17
図 12 ルーチンの新規作成 (Hello World!)	18
図 13 Hello world の作成	19
図 14 Hello world 引数の追加	20
図 15 クラス定義 新規作成	22
図 16 クラス定義の新規作成 (新規クラスウィザード)	23
図 17 クラス定義の作成 (新規クラスウィザード 2 画面目:クラスタイプ)	24
図 18 プロパティの追加 (新規プロパティウィザード)	25
図 19 スタジオのインスペクタ (プロパティのパラメーターの変更)	27
図 20 メソッドの追加 (printGender())	30
図 21 シリアルクラスの作成 (新規クラスウィザード 2 画面目)	34
図 22 FS.Addressのクラス定義全体	34
図 23 FS.Personテーブルのデータ参照	39
図 24 SqlComputeCodeの記述例	41
図 25 MS-Accessでのテーブルインポート	42
図 26 MS-Accessでのテーブル表示 (FS.Person)	43
図 27 クラスクエリの追加 (引数の設定)	44
図 28 クラスクエリ 選択カラムの設定	45
図 29 新規クエリウィザード:条件設定の画面	46
図 30 新規メソッドウィザード	47
図 31 新規メソッドウィザード 引数の指定	48
図 32 新規メソッドウィザード クラスメソッドの設定	48
図 33 printByName()のコード:ByNameクエリの実行	49

1. はじめに

ようこそ Caché の世界へ！

本ガイドは、Caché を初めて操作する方を対象に、Caché でのプログラミングの基本を解説します。本ガイドでは、ユーザの皆さんが Caché での開発を始められるのに必要な事項を Step-by-Step 形式で説明していきます。評価版の Caché を実際にインストールして操作しながらお読みいただければ、理解が深まると思います。

なお、本ガイドには、インストールの解説は含まれていません。インストール方法解説については、弊社WebサイトにあるCachéドキュメントをご参照ください。

http://docs.intersystems.com/cache_latestj/csp/docbook/DocBook.UI.Page.cls?KEY=GCI

評価版をご利用の方は、評価版ダウンロードサイト内にインストール解説書をご用意しております。

ファイル名: Cache2012.2PCKitForWindowsInstallationJPN.pdf

本ガイドをやり終えれば、Caché について開発をスタートする必要最小限の知識が得られます。さらに情報が必要な方は、オンライン・ドキュメントをご参照ください。

それでは、Caché をインストールすると標準機能として提供されるユーティリティについての説明から、始めていきましょう。実際に Caché をインストールしていただき、手を動かしながら、Caché に触れてみてください！

早速プログラミングを始めたい方は、4 章に進んでください。

＜製品版キットをご利用のお客様へ＞

（評価版をご利用のお客様は以下初期セキュリティ設定についてインストール時に選択できませんので、お読みいただかなくて結構です。）

本ガイドには、インストール時に設定する初期セキュリティ設定で「最小」を選択した状態での内容を記載しています。

インストール時初期セキュリティ設定の種類については、最小／標準／ロックダウン が選択できます。

初期セキュリティ設定詳細は以下 URL をご参照ください。

http://docs.intersystems.com/cache_latestj/csp/docbook/DocBook.UI.Page.cls?KEY=GCI_security#GCI_security_init_settings

セキュリティ全体の詳細説明については以下 URL をご参照ください。

http://docs.intersystems.com/cache_latestj/csp/docbook/DocBook.UI.Page.cls?KEY=GCAS

2. ユーティリティ

ここでは、Caché のユーティリティについて説明します。

Caché をインストールすると、Caché を操作するために必要なユーティリティが、標準でインストールされます。ユーティリティ起動方法は、Caché インストール後、Windows のタスクバーに現れる Caché ランチャーをクリックし、現れるメニューから起動します。

これらユーティリティは、Windows 上でのみ、ご利用いただけます。Caché を Windows 以外の OS へインストールしたときは、Caché クライアント機能を、Windows 上にインストールしていただき、指定の Caché サーバへ接続し、これらユーティリティをお使いいただけます。なお、管理ポータルは Web ベースで動作するユーティリティとなるため、Windows 以外でも利用可能です。Caché クライアントインストール方法、Caché サーバへの接続設定については、以下ドキュメントをご参照ください。

Caché クライアントインストールについて：

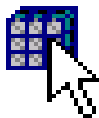
http://docs.intersystems.com/cache_latestj/csp/docbook/DocBook.UI.Page.cls?KEY=GCI_windows#GCI_windows_client

Caché サーバへの接続設定について：

http://docs.intersystems.com/cache_latestj/csp/docbook/DocBook.UI.Page.cls?KEY=GSA_connect_remote#GSA_connect_remote_define

では、ユーティリティを起動してみましょう！

- ① まず Caché ランチャーをクリックまたは、
右クリックします。



- ② 現れたメニューから、ユーティリティを選択します。



図 1 Caché ランチャーのメニュー

2.1. ドキュメント

Caché ランチャーから Caché のドキュメントを表示させることができます。

本ガイド内に、ドキュメントへのリンクがいくつかありますが、この説明以降のドキュメントについては、Caché ランチャーから起動するドキュメントへのリンクでご紹介しています。

また、Caché のドキュメントは、Caché の独自の Web 技術である Caché Server Page(CSP)を使用しています。Caché インストール時に、ネームスペース:DOCBOOK 内に、ドキュメントに必要なデータ類(クラス定義、CSP ファイルなど)、を用意しています。

2.2. スタジオ

Caché の統合開発環境であるスタジオでは、以下の開発環境を提供しています。

- ルーチンの作成
Caché 独自の Script 記述言語である以下、2 つのコード専用エディタ。
(Caché ObjectScript と Caché Basic)
- クラス定義の作成
ウィザードにより各クラスの属性を作成することができます。またインスペクタを利用して簡単に各属性について編集することができます。
- CSP ファイルの作成
CSP タグの挿入をツールバーからウィザードを使い挿入することができます。
- Web Service 用クラス定義の作成
WebService で必要となるクラス定義を、ウィザードを使い作成することができます。
- その他
SOAP ウィザードや XML スキーマウィザードの用意もあります。(「ツール」->「アドイン」のメニューよりウィザードを起動できます。)

また、スタジオでは、プロジェクト単位でアプリケーションに必要なソースをまとめて保存することができます。さらに、ソースコードの記述だけでなく、全てのタイプのソースコードをエクスポート／インポートすることもできます。プロジェクト単位でのエクスポート／インポートもできるため、別環境へのソースコードの移動も容易に行えます。

詳細は、ドキュメントをご覧ください。

(<http://localhost:57772/csp/docbook/DocBook.UI.Page.cls?KEY=GSTD>)

2.3. ターミナル

Caché の端末エミュレータとなります。ターミナルでは、Caché ObjectScript のコマンド、システムユーティリティなどを実行できます。アプリケーションを作成する際、作成したルーチンの動作確認などにも使用できます。

```
USER>
USER>write !,"今日は",$ZDATE($HOROLOG),"です",!
今日は11/08/2012です
USER>write $ZV
Cache for Windows (x86-64) 2012.2.1 (Build 705U) Wed Oct 24 2012 14:28:53 EDT
USER>write $SYSTEM.Version.GetOS()
Windows
USER>write $SYSTEM.Util.ManagerDirectory()
c:\¥intersystems¥cache¥mgr¥
USER>
```

2.4. 管理ポータル

Cachéの統合管理環境である管理ポータルでは、大きく分けて以下3つの機能を提供しています。管理ポータル画面について詳細は以下ドキュメントをご参照ください。

http://localhost:57772/csp/docbook/DocBook.UI.Page.cls?KEY=GSA_using_port al

([ドキュメント]->[Caché システム管理]->[Caché システム管理ガイド]-> チャプター1:管理ポータルの使用)

画面の起動には、Caché ランチャーのメニューを選択されるか以下URLを起動して下さい。

<http://localhost:57772/csp/sys/UtilHome.csp>

- システム管理(システム管理者タスク)



図 2 管理ポータル:システム管理

システムの構成情報(ネームスペース・データベースの作成、ECP の設定、起動環境調整用項目など)、セキュリティ管理、ライセンス管理、データベースの暗号化 を用意しています。詳細については以下ドキュメントをご参照ください。

構成について

http://localhost:57772/csp/docbook/DocBook.UI.Page.cls?KEY=GSA_config

セキュリティについて

http://localhost:57772/csp/docbook/DocBook.UI.Page.cls?KEY=GCAS_secmgmt

ライセンスについて

http://localhost:57772/csp/docbook/DocBook.UI.Page.cls?KEY=GSA_license

データベースの暗号化について

http://localhost:57772/csp/docbook/DocBook.UI.Page.cls?KEY=GCAS_encrypt

- システムエクスプローラー(データ管理)



図 3 管理ポータル:システムエクスプローラー

システムエクスプローラーには、データ管理用に、クラス定義、ルーチン、グローバル変数の一覧やインポート／エクスポート機能、SQL 文操作メニューやテーブルなど用意しています。

SQL の項目では、Cache クラス定義により定義された Cache のデータをテーブル形式で参照することができます。また SQL 文の実行、ASCII ファイルからのデータインポートやエクスポート機能など取り揃えています。

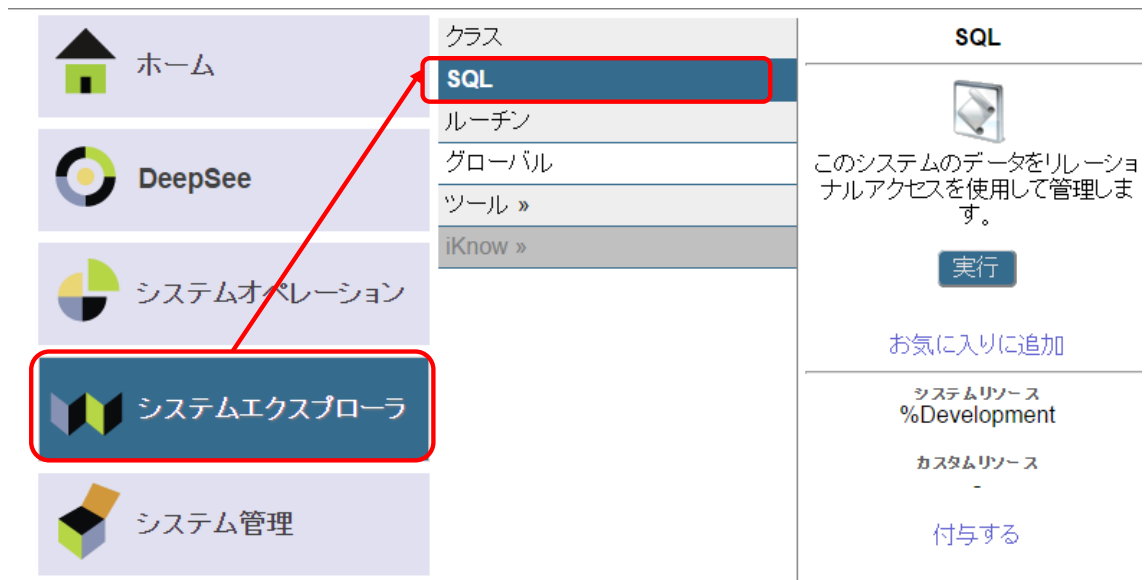


図 4 管理ポータル:システムエクスプローラ:SQL

その他機能では、実行したSQLクエリに対してのクエリプラン¹ の表示機能もあります。

¹ クエリプランとは、特定のクエリが実行されたときにそれがデータベース内からどのようにデータを取得するか(プラン)の表示とそれに要する相対コストを概算するものです。クエリプランを使用し、あるテーブルに対する特定のクエリを効率よく実行するために、どのようなインデックスを設定すればよいかなど、判断基準となる数値を取得することができます。

クエリプランを表示するまでの手順は以下のとおりです。

1. 管理ポータル→SQL から SQL 文実行画面を起動し、SELECT 文を記入します。
2. 「クエリ実行」ボタン押下でクエリを実行します。
3. クエリプランを照る場合は、クエリ実行ボタンの隣にあるボタン「クエリプラン表示」ボタン押下します。

The screenshot illustrates the steps to view a query plan in the InterSystems Caché management portal. The interface is divided into a left sidebar with navigation links (Home, DeepSee, System Explorer, System Management) and a main content area. The 'SQL' tab is selected in the sidebar (labeled ②). The main area shows the 'SQL' execution screen with a text input field containing a SELECT query. A red box labeled ③ points to the 'NameSpace' dropdown menu, which is set to '%SYS'. A red box labeled ④ points to the 'SAMPLES' database selected in the 'NameSpace Selection' dialog. Below the query input, the '実行' (Execute) button is highlighted, with a red box labeled ⑤ pointing to it. A red box labeled ⑥ points to the 'クエリプラン表示' (Query Plan) button next to the '実行' button. The output area shows the 'クエリ文字列' (Query Text) and the 'クエリプラン' (Query Plan) details, including the relative cost (2067.4) and the execution plan steps. A red box labeled ⑦ points to the 'クエリプラン表示' button in the output area.

② SQL

③ ネームスペースを確認します。ランチャーから管理ポータルを開いた場合は%SYS ネームスペースが選択されています。

④

⑤ クエリ実行タブでSQL文を指定し「実行」ボタンを押下します。

⑥ クエリ実行後に「プラン表示」ボタンを押下すると、結果表示画面が「クエリプラン」の表示に切り替わります。

⑦

クエリ文字列

```
select Name,Home_Zip from Sample.Person where Home_Zip %startswith 7
```

クエリプラン

相対コスト = 2067.4

- Call [module B](#), which populates bitmap temp-file A.
- Read bitmap temp-file A, looping on ID.
- For each row:
 - Read master map Sample.Person.IDKEY, using the given idkey value. Output the row.

module B

- Read bitmap index Sample.Person.ZipCode, looping on %SQLUPPER(Home_Zip) (with a %STARTSWITH range condition) and bitmap chunks.
- For each bitmap chunk:
 - OR the bitmap chunk into bitmap temp-file A.

図 5 管理ポータル:クエリプランの表示

- システムオペレーション(システム運用タスク)



図 6 管理ポータル:システムオペレーション

システムオペレーションでは、システムの運用管理面で利用するタスク(バックアップ、ジャーナル管理、ロック管理など)を用意しています。

各項目について詳細はドキュメントの「Caché監視ガイド」、「Caché高可用性ガイド」、「Cachéシステム管理ガイド」をご参照ください。

<http://localhost:57772/csp/docbook/DocBook.UI.Page.cls?KEY=SETSysAdmin>

3. データベースとネームスペース

ここでは、Cache のデータへアクセスするために、指定する「ネームスペース」と、ネームスペースから指定される「データベース」についての概念を説明します。

まず、Cache の中でのネームスペースとデータベースとは、

＜ネームスペース＞

ネームスペースとは、ユーザからみた「作業領域」になります。データにアクセスするとき、クラス定義／ルーチンなどを作成するとき、プログラムを実行するとき、ODBC で接続するとき、CSP アプリケーションを動かすとき、まずは、ネームスペースを指定するところから、作業ははじまります。

＜データベース＞

データベースとは、データ、ルーチン、クラス定義などが格納されている場所です。この場所にアクセスするためには、ユーザはネームスペースを指定し、アクセスします。

では、ネームスペースとデータベースの関係は？ については

ネームスペースから、アクセスしたいデータベースを指定する。(ネームスペースからデータベースをマッピングする)

となります。では、どこでその指定を設定するか？については、管理ポータルから行います。

(1) 管理ポータル

 ホーム  DeepSee  システムオペレーション  システムエクスプローラ  システム管理	構成 »	システム構成 »	メモリと開始設定
	セキュリティ »	接続性 »	ネームスペース
	ライセンス »	ミラー設定 »	ローカルデータベース
	暗号化 »	データベースバックアップ »	リモートデータベース
	エンタープライズ・マネージャ	CSPゲートウェイ管理	ジャーナル設定
		SQL とオブジェクトの設定 »	
		デバイス設定 »	
		国際言語設定 »	
		Zen レポート »	
		追加の設定 »	

図 7 管理ポータル:構成→システム構成

管理ポータル→「システム管理」→「構成」→「システム構成」→「ネームスペース」へ移動します。「ネームスペース」のメニュー名をクリックするか、移動ボタンを押下すると、ネームスペース一覧画面へ移動します。

(2) ネームスペース

対象となるネームスペース名のリンクを押下します。

ネームスペース

サーバ: **iiijima-LetsNote** ネームスペース: **%SYS**
ユーザ: **UnknownUser** ライセンス先: **ISC Learning Services**

新規ネームスペース作成

現在のネームスペースおよびそれらのグローバル/ルーチンに対するデフォルトデータベース::

フィルタ: ページサイズ: 0 最大行数: 1000 結果: 4 ページ: 1 の 1

Namespace	Globals	Routines	Temp Storage				
%SYS	CACHESYS	CACHESYS	CACHETEMP	グローバルマッピング	ルーチンマッピング	パッケージマッピング	削除
DOCBOOK	DOCBOOK	DOCBOOK	CACHETEMP	グローバルマッピング	ルーチンマッピング	パッケージマッピング	削除
SAMPLES	SAMPLES	SAMPLES	CACHETEMP	グローバルマッピング	ルーチンマッピング	パッケージマッピング	削除
USER	USER	USER	CACHETEMP	グローバルマッピング	ルーチンマッピング	パッケージマッピング	削除

図 8 ネームスペース一覧

USER ネームスペースの 編集 メニューを選択します。

編集: **USER**

サーバ: **iiijima-LetsNote** ネ
ユーザ: **UnknownUser** ラ

保存 キャンセル

ネームスペース USER の設定を編集

名前	USER	USERネームスペースには、USERデータベースが設定されています。
グローバルのデフォルトデータベース	USER	
ルーチンのデフォルトデータベース	USER	
仮ストレージのデフォルトデータベース	CACHETEMP	

図 9 ネームスペースのデフォルトデータベースの確認

ネームスペースでは、デフォルトでアクセスするデータベースを指定します。

(図 9 ネームスペースのデフォルトデータベースの確認 の グローバルのデフォルトデータベース、ルーチンのデフォルトデータベースには、ユーザ定義のデータベースが指定されています。)

ユーザ定義のデータベースはデフォルトでは1つのネームスペースに対して、1つしか指定できませんが、別のデータベース上に存在するデータまたはルーチンを個々にマッピングすることもできます。(P15 図 8 ネームスペース一覧 の画面にある「グローバルマッピング」で定義します。)

このように、ネームスペースに全て指定することもできますし、勿論、アプリケーションプログラムの中でネームスペースの指定を変更することもできます。

ネームスペースでは、ユーザが操作したいデータベース、または指定したデフォルトデータベース以外に存在するデータベースのデータ／ルーチン／パッケージへのアクセスを定義したものです。

マッピングを利用することで、デフォルトデータベース以外の複数のデータベースに存在する情報を、あたかも 1 つのデータベースのようなイメージでアクセスできます。

言い換えてみれば、(正確な言葉ではありませんが)、ユーザから見ると

ネームスペース＝データベース

と考えてもよいかもしれません。

たとえば、以下のような使い方もできます。

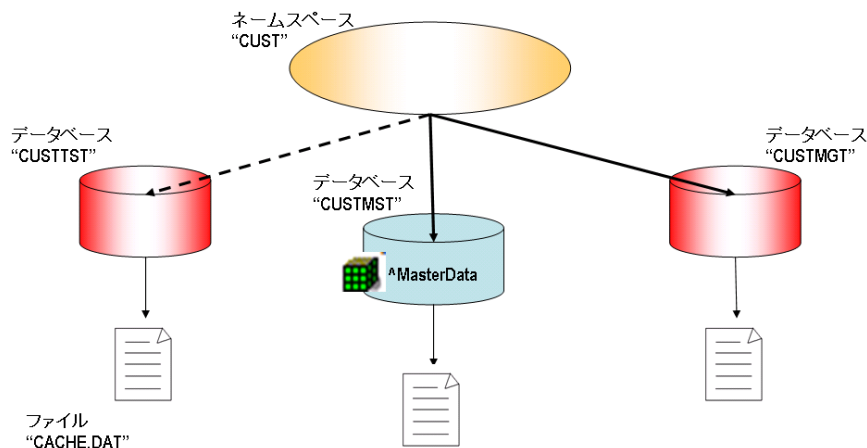
テスト環境用データベース「CUSTTST」、本番環境用データベース「CUSTMGT」、(円柱の図)があるとして、ここに、アプリケーションで使用する固定データが、データベース「CUSTMST」が格納されているとします。(固定データ= ^MasterData)

この場合、ネームスペース「CUST」では、以下のように、接続データベースを指定できます。

デフォルトデータベース: グローバル ⇒ 「CUSTTST」または「CUSTMGT」

グローバルマッピング ⇒ 「CUSTMST」の ^MasterData

実際アプリケーションを作成するとき、テスト環境であっても、本番環境であっても、ユーザはアクセスするデータベースの位置を意識しなくても、ネームスペースの名前だけを知っていれば、データベースへ接続できます。また、接続先データベースの変更も、アプリケーションロジックの変更なしに、変更することができます。



(ファイル「CACHE.DAT」については、データベースの「物理ファイル」となります。これは、データベース毎に必ず1つ作成されます。)

図 10 ネームスペースとデータベース 定義例

4. 最初の Cache プログラム

ここからは、Cache のプログラミングについて説明していきます。なお、特に断りのない限りネームスペース"USER"を使用しているものとして説明しています。また、各章の区切りごとに Cache スタジオにインポート可能な形式でのサンプルを用意していますので、ご活用ください。

(インストール後初めてCacheスタジオを起動した場合、ログイン画面が表示されます。ユーザ名、パスワードは未記入のままログインしてください。次回からログイン画面は表示されません。)

Cache はデータベースであると同時に、強力な開発用スクリプト言語を装備しています。このスクリプト言語を Cache ObjectScriptと呼んでいます。Cache ObjectScript はオブジェクト指向プログラミングをサポートしており、また Cache データベースへの SQL アクセス、ダイレクトアクセスを行うことができます。また、文字列処理などのユーティリティ関数を豊富に用意しています。

4.1. Hello world

では、最初の Cache のプログラムということで、典型的な例として Hello World プログラムを書いてみましょう。これは、実行すると単に Hello World!と表示するだけのプログラムです。

Cache スタジオで、新しくルーチンを作成します。作成するには、スタジオのメニューから、「ファイル」→「新規作成」→「一般」→「Cache ObjectScript ルーチン」を選択します。

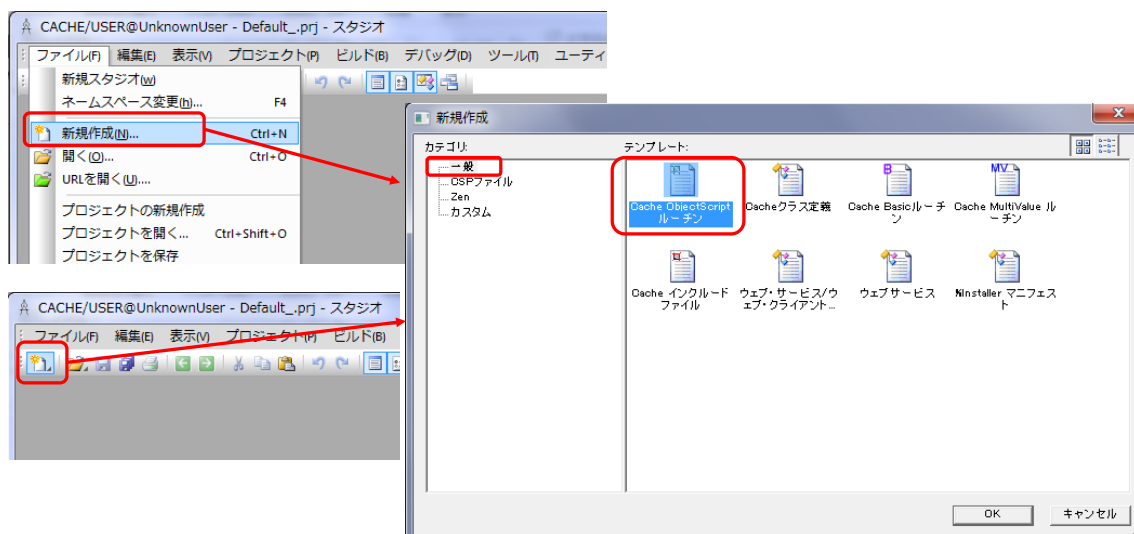


図 11 ルーチンの新規作成(Hello World!)

エディタが開くと、コードウインドウに白紙の領域が表示されますので、ここにコードを記述していきます。

次のようなコードを記述してください。

```
Hello() public {  
    write "Hello, world!"  
}
```

図 12 Hello world の作成

ここで"Hello"は、ルーチンのラベル(もしくはエントリ)と呼ばれ、Public 宣言により外部から呼び出し可能なことを示しています。また、Write は、Cache ObjectScript のコマンドの一つで端末に対して出力を行うコマンドです。なおコマンドを書き始める場合には先頭にスペースまたはタブを書く必要があります。

また、Cache では、Write などのコマンドは大文字小文字の区別をしません、変数名やラベル名では大文字小文字が区別されますので、注意してください。

さて、このルーチンを実行するわけですが、その前にルーチンを保存・コンパイルする必要があります。これを行うには、スタジオで「ビルド」メニューの「コンパイル」を選択します。ルーチンを保存せずにコンパイルすると、まず保存のダイアログボックスが出ますので、そこで、ルーチンの名前を入力します。ここでは、First.mac とします(.mac とは Cache のルーチンの形式を示す拡張子です)。「OK」ボタンを押すと、コンパイルが行われ、成功すると、スタジオの出力ウインドウ(下部ウインドウ)に、

```
03/25/2015 15:32:07 に修飾子 'cukb /checkuptodate=expandedonly' でコンパイルを開始しました。  
ルーチンのコンパイル中 : First.MAC  
コンパイルが正常に終了しました (所要時間: 0.037 秒)。
```

上記のように、コンパイル結果が出力されます。

では、いよいよ実行です。Cache には、対話的にルーチンを実行するための Cache ターミナルが用意されています。Cache ランチャーからターミナルを起動します。

ターミナルが起動すると、プロンプトにネームスペースが表示されます（通常は USER）。
そのプロンプトから、

Do Hello^First()

と入力し、リターンキーを押下します。これは、ルーチン First の Hello ラベル（エントリ）を実行しなさいという Cache のコマンドの形式です。実行すると、次のような結果が表示されるはずです。

Hello, world!

4.2. パラメータを渡す

もちろん、ルーチンのエントリに対して引数を定義し、実行時に値を渡すことも出来ます。
Hello() エントリの定義を次のように変更してみましょう。

```
Hello(name) Public {  
    Write "Hello, world, "_name_"!"  
}
```

図 13 Hello world 引数の追加

ここで、“_” が文字列の結合演算子として使用されていることに注意してください。
そしてコンパイル後、実行します。例えば、

Do Hello^First(“太郎さん”)

と入力すれば、

Hello, world, 太郎さん!

と出力されます。

ここまでの作業のサンプルは、Step1 フォルダにあります。

5. Cache オブジェクト

これまでで、最初のプログラム Hello を見てきましたが、そこではオブジェクト指向の考え方は使用していませんでした。しかし、最初に説明しましたように、Cache に装備された言語、Cache ObjectScript は、完全なオブジェクト指向をサポートしています。ここからは、オブジェクト指向の考え方を使ったプログラムの例を見ていきます。

5.1. クラスを作成

オブジェクト指向では、クラスがプログラムの単位となります。詳細はオブジェクト指向の入門書などを参照してください。

では、Cache スタジオでクラスを作成してみましょう。

ここでは、「人」を表す Person クラスを定義してみます。クラスを定義する際には、そのクラスが持つ属性を定義する必要があります。Cache ではこの属性のことをプロパティと呼んでいます。では、「人」が持つプロパティは何でしょうか。もちろんこれは、開発するシステムによって色々なものが考えられますが、ここでは、次のようなプロパティを考えてみます。

《永続クラス》 Person
Name As %String Address As %String Gender As %String DOB As %Date

Name は人の名前、Address は住所、Gender は性別、DOB は誕生日 として定義します。

Caché スタジオでクラスを定義するには、「ファイル」→「新規作成」を選択します。
表示されたダイアログの左画面で「一般」を選び、右画面で「Cache クラス定義」を選び「OK」ボタンを押下します。

以下、ウィザード形式でクラス定義の作成画面が始まります。

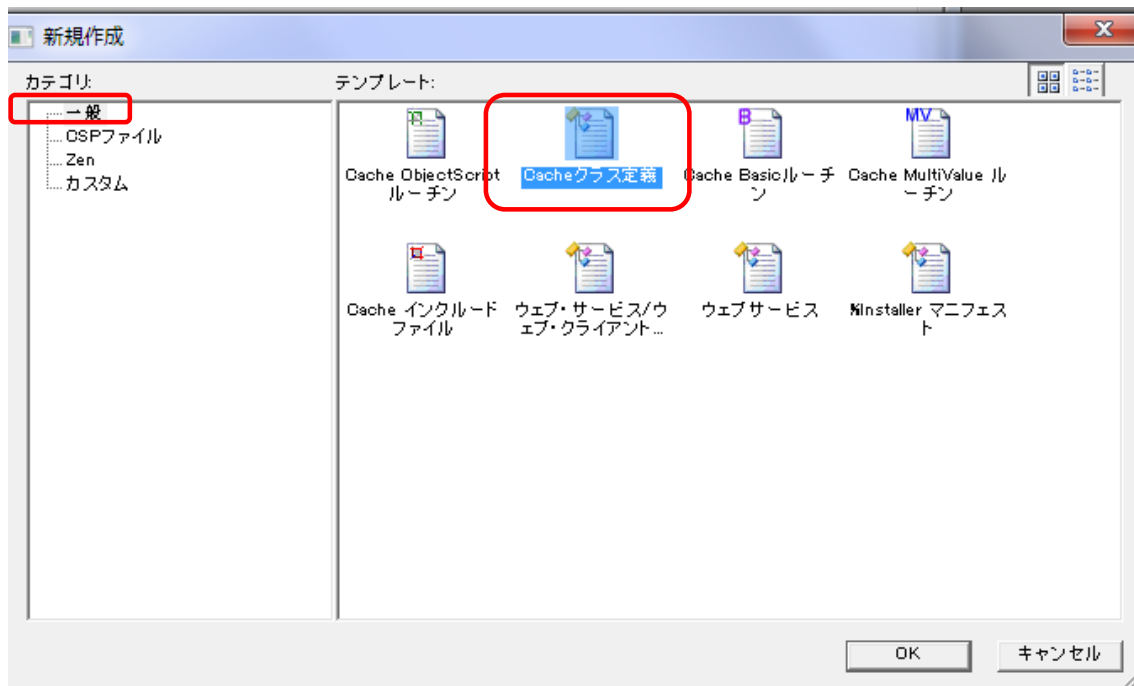


図 14 クラス定義 新規作成

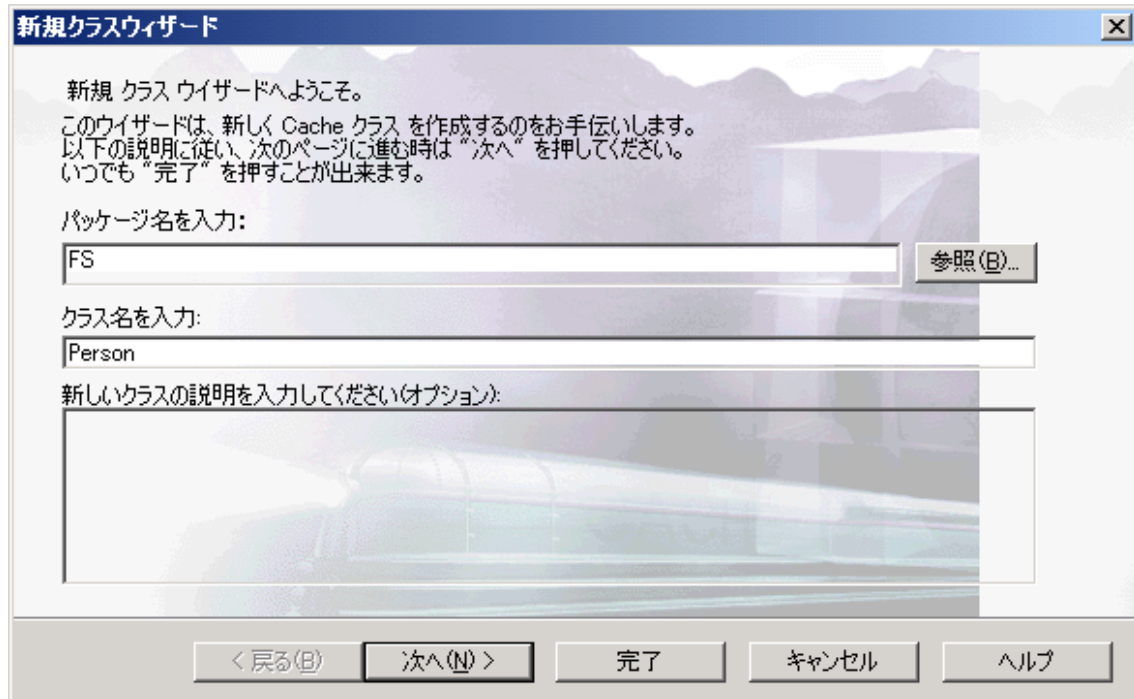


図 15 クラス定義の新規作成(新規クラスウィザード)

まずは、パッケージ名とクラス名を入力します。パッケージとは、クラスを分類する単位です。Java のようにパッケージを階層的にすることも可能です。ここでは、FS (First Step の略) をパッケージとし、Person というクラス名にします。(この場合、このクラスの完全指定は、FS.Person と、ドットを使用してパッケージとクラスを指定します)。

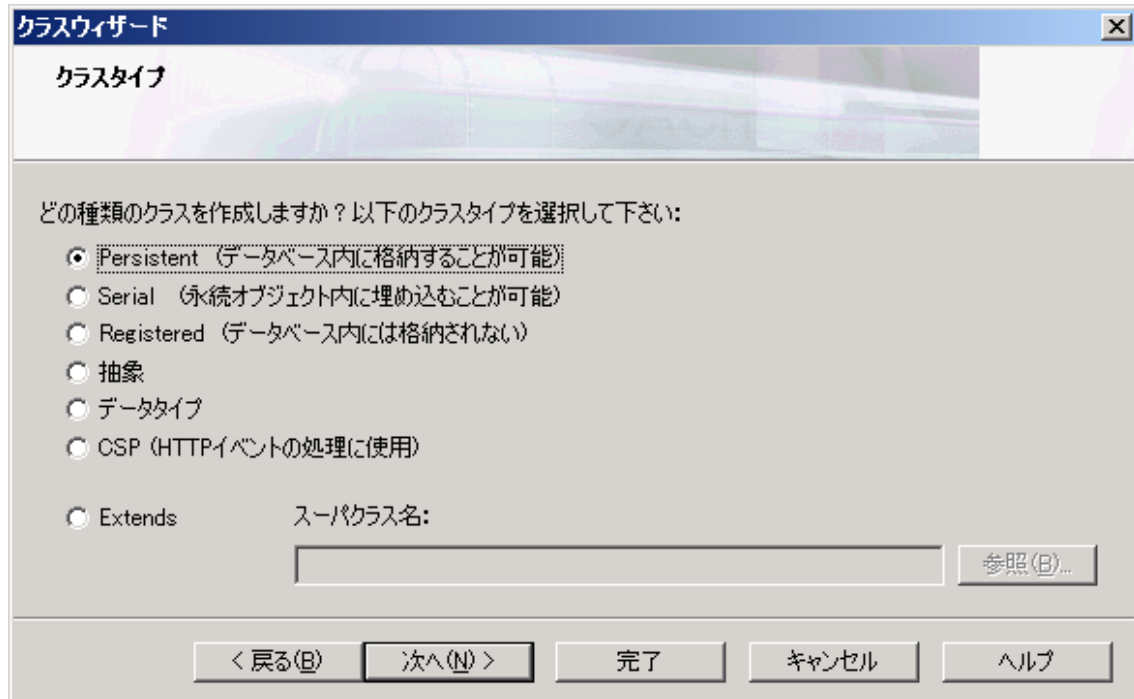


図 16 クラス定義の作成(新規クラスウィザード 2 画面目:クラスタイプ)

次に進むと、クラスのタイプを指定する画面に移ります。Cache にはいくつかのタイプのクラスが定義できますが、ここでは「Persistent」を選択します。これは、データベースに保存することができるクラスで、これにより、このクラスのインスタンスをデータベースに保存するコードが自動的に生成されます。

ウィザードでは次に進んでその他の項目を設定することもできますが、ここでは「完了」をクリックし、一旦クラスの定義を終了します。そうすると、コードウインドウにクラス定義のテキストが表示されます。

5.2. プロパティの定義

それでは、プロパティの定義に移りましょう。

まずは、ウィザード形式で名前プロパティの定義を行ってみます。

スタジオのメニューで、「クラス」→「追加」→「プロパティ」を選択します。

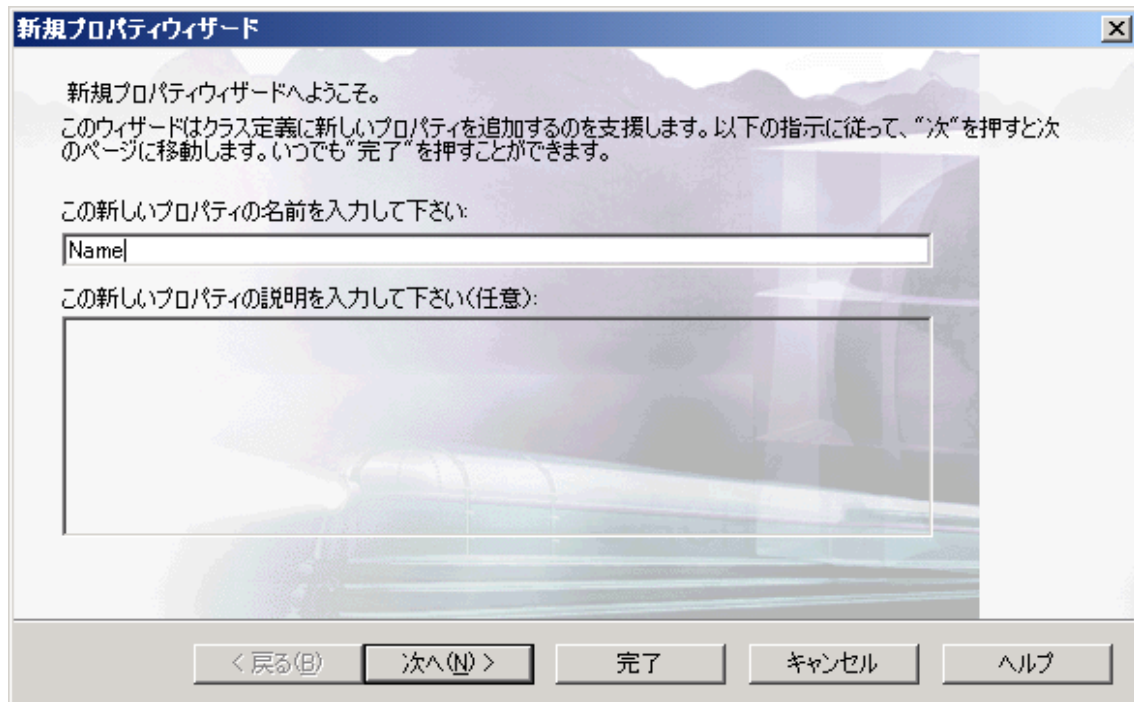


図 17 プロパティの追加(新規プロパティウィザード)

まずは名前を入力します(ここでは Name)。次へ進むとプロパティの型を指定する画面になりますので、ここでは文字列を表す %String とします(デフォルトで %String になっていますのでそのままにしてください)。

そして、「完了」をクリックします(ウィザードは次へ進むこともできますが、ここではここで完了します)。そうすると、コードウインドウに、今定義した Name プロパティの定義テキストが追加されているのが分かります。

その他のプロパティもウィザードで定義することもできますが、もっと簡単な方法があります。それは、コードウィンドウを直接編集することです。コードウィンドウで編集を行い次のように残りのプロパティを定義してみてください。

```
Property Name As %String;  
Property Address As %String;  
Property Gender As %String;  
Property DOB As %Date;
```

ここで、誕生日 (DOB) プロパティの型が日付を表す `%Date` 型であることに注意してください。Gender (性別) は、値として "M" (男性) もしくは "F" (女性) の 2 通りを持つプロパティであるとしてします。このような場合、Caché では、値の選択肢をあらかじめ定義しておくことが可能です。これを行う一つの方法は、スタジオの左にあるインスペクタを利用することです。インスペクタの左上のコンボボックスで、"Property" を選択すると定義されているプロパティの一覧が表示されます。ここで、"Gender" の部分をダブルクリックすると、Gender プロパティの属性が詳細に表示されます。そして、"Parameter" の "+" をクリックし展開すると、設定可能なパラメータが展開表示されます。

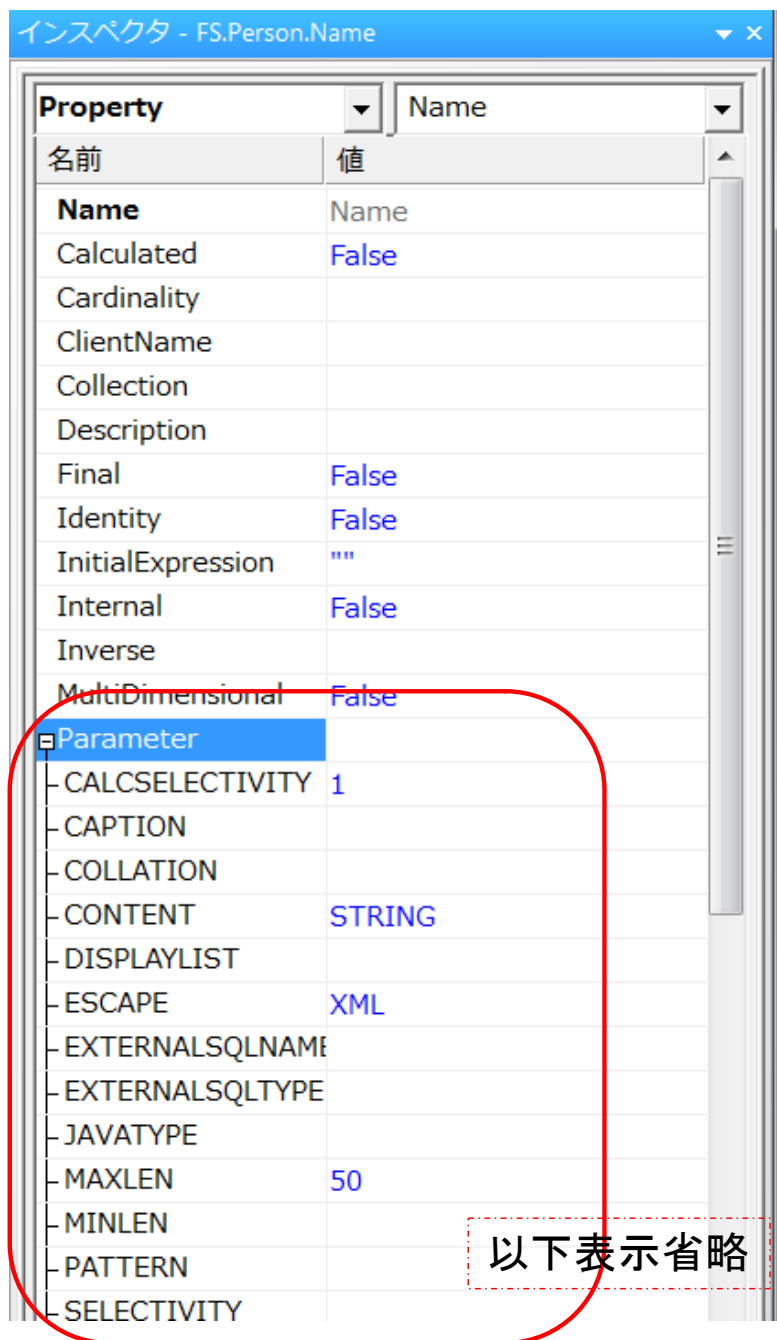


図 18 スタジオのインスペクタ(プロパティのパラメーターの変更)

ここで、VALUELIST パラメータに ",M,F" と入力します。(", "自体は入力しないで下さい)。そして、コードウインドウの部分をクリックすると、

```
Property Gender As %String(VALUELIST = ",M,F");
```

のように、インスペクタで定義した VALUELIST パラメータがコードにも反映されることを確認してく

ださい。なお、VALUELIST にセットするとき、最初の文字をデリミタとみなされることに注意してください。、M,F としてもいいですし、例えば #M#F としても同じ結果になります。ですが、最初のデリミタ付け忘れて、M,F などとすると、M をデリミタとみなしてしまいますので注意してください。

5.3. コンパイル

それではクラスをコンパイルします(スタジオのメニュー「ビルド」→「コンパイル」)。コンパイルが成功すれば、Caché はこの FS.Person クラスのインスタンスをデータベースに格納するコードを生成します。たったこれだけで、オブジェクトを処理するデータベースが構築できるのです！

5.4. インスタンスの生成・保存

では早速作成した FS.Person クラスを使用して、そのインスタンスを生成してみましょう。Caché ObjectScript は即時実行可能な言語ですので、このようなアドホックな操作も簡単に実行できます。では、Caché ターミナルを開いて次のようなコマンドを実行してください。

```
Set p = ##class(FS.Person).%New()
```

このコマンドでは、FS.Person クラスのインスタンスを生成し、メモリ上の変数 p に格納しています。この時点では、インスタンスはまだデータベースに保存されていないことに注意してください。また、%New() は新しいインスタンスを生成するクラスメソッドで、クラスメソッドを呼び出すには、##class(classname).method() というシンタックスを使用します。

このインスタンスに対して、各プロパティの値を設定します。

```
Set p.Name = "インターシステムズ 太郎"  
Set p.Address = "東京都新宿区西新宿"  
Set p.Gender = "M"  
Set p.DOB = ##class(%Date).DisplayToLogical("5/1/1978")
```

DOB プロパティ設定時に、日付を##class(%Date).DisplayToLogical()によって、文字列から Caché の内部日付フォーマットに変換していることにも注意してください。

最後に、

Write p.% Save()

としてメモリ上のインスタンスをデータベースに保存します。1 が出力されれば保存が成功したことを示します。

以上でインスタンスが1つデータベースに保存されました。非常に簡単ですね。

5.5. インスタンスの読み込み

本当にインスタンスがデータベースに保存されたか、別のターミナルを開いてそこからインスタンスを読み込むことによって確かめてみましょう。

別の Caché ターミナルから次のコマンドを実行します。

Set p = ##class(FS.Person).%OpenId(1)

ここで%OpenId()は、指定されたOID(Object ID)を持つインスタンスをオープンするメソッドです。OID は、インスタンス保存時に Caché により自動的に付与され、通常は正の整数です。ですから、クラスの最初のインスタンスは通常 OID=1 となるので、このコマンドで先程保存されたインスタンスを読み込むことができる訳です。もちろん、OID を指定せずにインスタンスを検索する方法もあります。これについては後のセクションで説明します。

オープンしたインスタンスのプロパティの値を確認してみます。

Write p.Name

インターシステムズ 太郎

Write p.Address

東京都新宿区西新宿

先程保存した値が出力されることを確認してください。

5.6. メソッドの定義

オブジェクト指向では、プロパティと共に、クラスを構成するもう一つの要素はメソッドです。メソッドは、クラスのインスタンスに対して行われる操作を記述するプログラムのことです。例えば、注文オブジェクトに対して合計を求めたり、明細を追加したりするなどのメソッドが考えられます。

```
Method printGender()  
{  
    If ..Gender = "M" {  
        Write "私は男です"  
    }  
    Else {  
        Write "私は女です"  
    }  
}
```

図 19 メソッドの追加(printGender())

ここでは、簡単な例として、男性か女性かをターミナルに出力するメソッドを考えてみます。Person クラス定義の中に、このメソッドを追加します。メソッドを定義するには、ウィザードを使用することもできますが、ここではコードウインドウを直接編集することになります。

メソッドは、Method 宣言によって定義を始めます。メソッド内で自インスタンスのプロパティや他のメソッドを参照する場合は、**..*property*** や **..*method()*** のように **“..”** をつけて、自インスタンスであることを示します。またこのメソッドは、Cache ObjectScript の if ... else 文の例になっていることにも注意してください。

では、このメソッドを先程作ったインスタンスに対して実行してみます。実行の前にコンパイルしてください。

注意

コンパイルする前に、Cache ターミナルでオープンしているインスタンスを全てクローズする必要があります。そのためには、Set p="" と変数を "" でクリアするか、Kill コマンドを実行し変数を削除します。

再び、Caché ターミナルで先程の %OpenId() を使用してインスタンスをオープンします。そのインスタンス p に対して、

Do p.printGender()

を実行すると、Gender プロパティの値に応じて端末に出力されるはずです。Do コマンドは、このように戻り値がないメソッドの実行にも使用します。

5.7. 計算プロパティ

FS.Person クラスに年齢を表す Age プロパティがないのを不思議に思われたかもしれません。が、考えてみると年齢は 1 年ごとに増えていきますからデータベースに年齢をもつのは効率的な方法とはいえません。とはいえ、年齢は利用者からすると、「人」が持っていて当然の属性であると思うでしょう。

Caché はそのような場合に最適な仕組みを持っています。それが「計算プロパティ」です。計算プロパティは、実際にデータベースには格納されませんが、参照時に計算してあたかもプロパティが存在するかのように見せる仕組みです。

では、実際に Age プロパティを定義してみましょう。スタジオのコードウインドウの任意の行に、次のような定義を加えます。

```
Property Age As %Integer [ Calculated, Readonly];
```

通常のプロパティと同様に定義しますが、最後に "Calculated" という修飾をつけて、計算プロパティであることを宣言しています。このように宣言されたプロパティは、実際にデータベースに格納されません。また、年齢は参照のみ可能ですので、"Readonly" も付けています。

では、参照時にどのような値を返せばよいのでしょうか。年齢の場合、現在日付と誕生日を比較して計算すれば良いでしょう。計算プロパティでは、このような計算ロジックを、ある特定の名前のメソッドで実装します。その名称は以下のとおりです。

```
PropertyGet()
```

作成中クラスでは、AgeGet() となります。では、AgeGet()の実装を見てみます。

```
Method AgeGet() As %Integer
{
    Quit (+$Horolog-..DOB)/365
}
```

ここでいくつかのことについて説明しなければなりません。まず、+\$Horolog という表現式ですが、これは Caché のシステム変数の一つで、現在日付を 1841 年 1 月 1 日 以来の日数を返します。(省略形では、\$H と記述できます。)また、DOB プロパティもこの形式で保存されていますので、"+\$Horolog-..DOB" という式は、誕生日から今日までの日数を求めていることとなります。これを 365 で割った商を計算して、簡便的に年齢を求めています('/' は商を表す演算子です)。

この値を Quit の引数にしています。Quit は、C や Java の return と同様、メソッドを終了し、戻り値があればそれを渡します。

以上で Age の定義ができましたので、いつものようにコンパイルして、OID=1 のインスタンスをオープンし、次のコマンドを実行してみます。

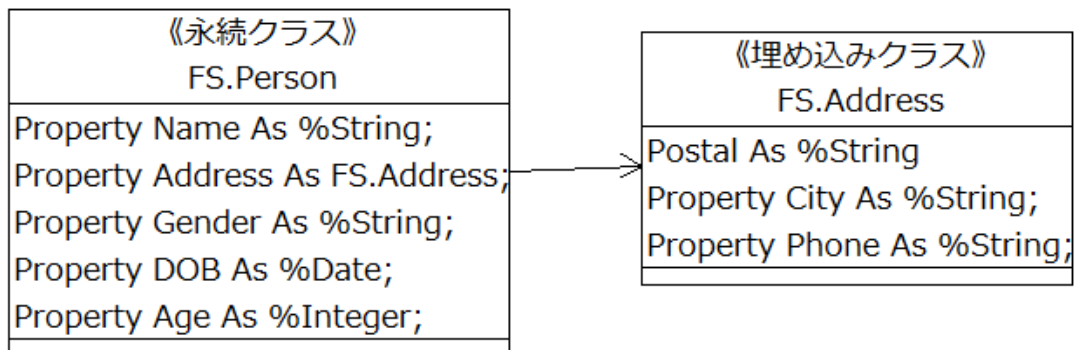
Write p.Age

ターミナルに年齢が表示されたと思います。

5.8. 埋め込みオブジェクト

これまでは、住所プロパティ(Address)は単なる文字列として定義していました。が、住所には、郵便番号や電話番号など付帯的な情報があるのが通常です。その場合、住所を別のクラスとして定義することが考えられます。ただ、住所クラスのインスタンスをそれ単独で存在させるのは、あまり意味がありませんし、システムのオーバヘッドの観点からも好ましくありません。

このような場合、Cache では埋め込みオブジェクトという仕組みが利用できます。まずは、住所クラス (FS.Address) を定義しましょう。



FS.Person と同様に、新規クラスを作成するウィザードで定義を行いますが、ウィザードの 2 ページ目のクラスタイプを選択するところで、“Serial”を選びます。こうすることで、FS.Address 自身はインスタンス化できませんが、他のオブジェクトのプロパティとして FS.Address インスタンスを埋め込むことが可能になります。

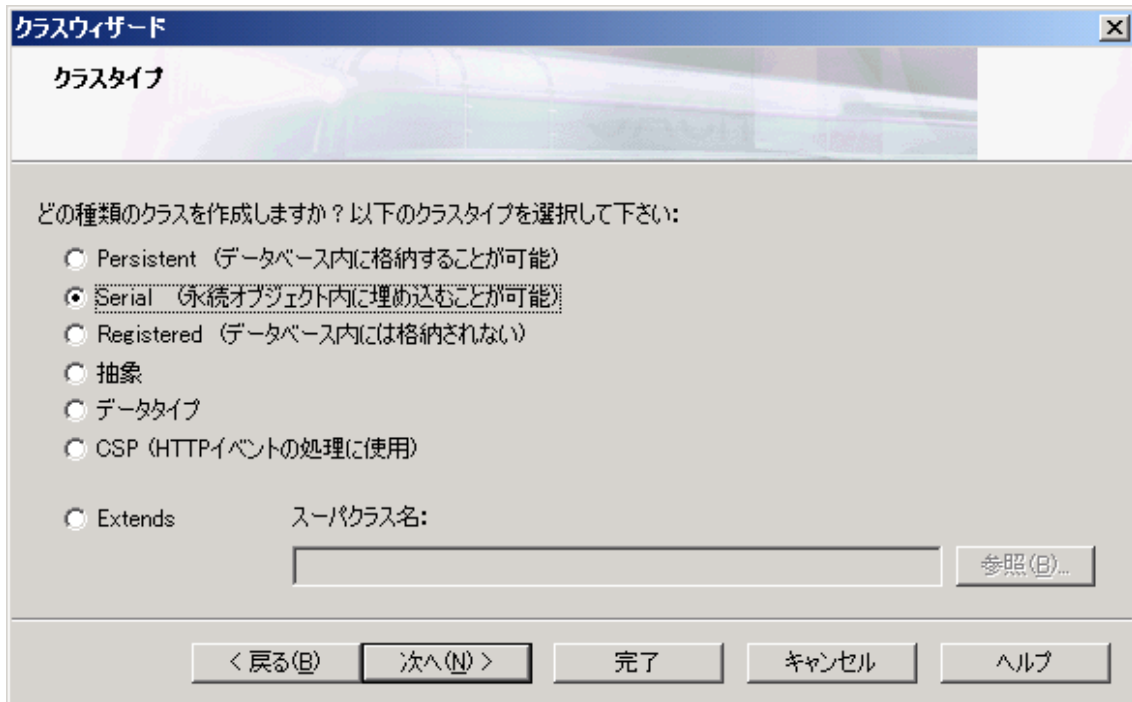


図 20 シリアルクラスの作成(新規クラスウィザード 2 画面目)

FS.Address の完全な定義は次のようになります。

```
Class FS.Address Extends %SerialObject
{
    Property Postal As %String;
    Property City As %String;
    Property Phone As %String;
}
```

図 21 FS.Address のクラス定義全体

次に、FS.Person の Address プロパティのデータ型を修正します。FS.Person の定義の Address の部分を次のように変更します(太字の部分の変更部分です)。

Property Address As **FS.Address**;

そして、FS.Person もコンパイルします。これで、クラス定義の修正は完了です。あとは、すでに存在する FS.Person のインスタンスを、埋め込みオブジェクトを利用する形に修正します。Caché ターミナルを開いて、次のコマンドを実行します。

注意)コンパイルする前に、Caché ターミナルでオープンしているインスタンスを全てクローズする必要があります。そのためには、Set p=""と変数を""でクリアするか、Kill コマンドを実行します。

```
Set p=##class(FS.Person).%OpenId(1) // 既に作成している 1 件目の Person をオープン
Set p.Address = ""
Set p.Address.Postal = "160-0023"
Set p.Address.City = "新宿区"
Set p.Address.Phone = "03-1234-5678"
Write p.%Save()
```

2 行目の代入文で、今までの文字列の値を削除し、その後の 3 つのコマンドで、埋め込みオブジェクト(p.Address.property)に値を設定しています。最後の %Save() メソッドで、インスタンスの上書き保存を行います。1 が出力されれば正常に保存が完了しました。

5.9. Populate 機能によるテストデータの生成

これまでは、ターミナルから手で1件のインスタンスを作成し、それを元に操作を行ってききましたが、テストとしてはもう少しテストデータが欲しいところです。

Caché ではこのような時に便利な機能があります。それは、"%Populate" ユーティリティクラスです。このクラスは、大量のテストデータを自動的に生成するロジックを実装したクラスで、今回の FS.Person, FS.Address も %Populate を継承することで自動生成の機能を使用することができます。（%Populate クラスのフルネームは %Library.Populate クラスです。）

%Populate を継承するには、各クラス定義でスタジオのインスペクタを利用すると便利です。

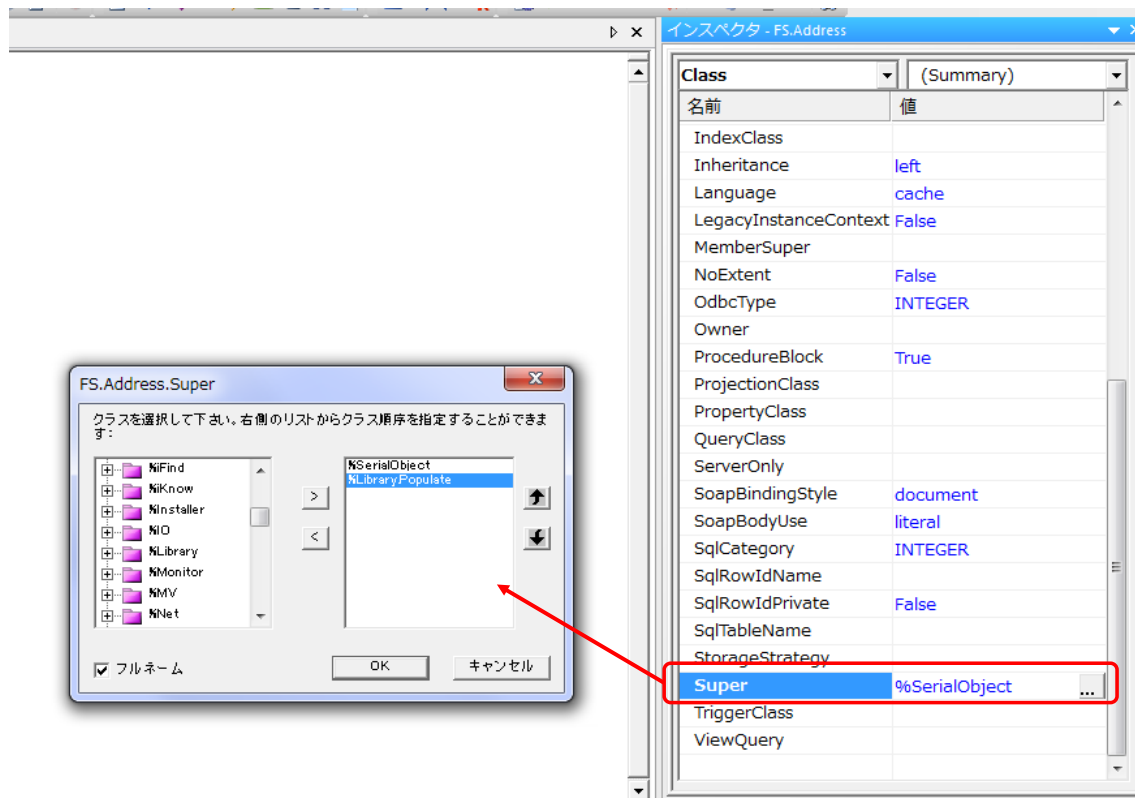


図 22 クラス定義:インスペクタの利用(スーパークラスの追加)

インスペクタでスーパークラスの指定を追加した後のコードウィンドウは、以下のとおりです。

```
Class FS.Person Extends ( %Persistent, %Library.Populate)
Class FS.Address Extends ( %SerialObject, %Library.Populate)
```

基本的にこれで²、英語のデータではありますが、自動生成ができるようになりますのですが、FS.Addressでは、Cityプロパティに対してのみ、%Populateがどのような値を生成するかのヒントを与える必要があります。そこで、FS.AddressのCityの定義を次のように変更します(ここでも太字に注目してください)。

```
Property City As %String(POPSPEC = "City()");
```

これで、%Populate は City に対して、市の名前を生成します(英語です)。

では、コンパイルして次のコマンドを実行します。

```
Write ##class(FS.Person).Populate(1000)
```

これはインスタンスを 1000 個作成するよう指示するコマンドです。そして、実際に作成された個数が返されます(通常は 1000 になるはずです)。

これで、先程のインスタンスと合わせて 1001 個のインスタンスがデータベースに存在しますので、ターミナルから、いろいろな OID のインスタンスをオープンして、そのプロパティの値を参照してみてください。

ここまでの作業のサンプルは、Step2 フォルダにあります。

² %Populate クラスの継承は、インスペクタを使用せずコードウィンドウに直接追記することも出来ます。

6. Caché SQL

これまでは、Caché でクラスを定義し、そのインスタンスを生成・アクセスすることを説明しました。Caché がオブジェクト指向をサポートし、しかも非常に簡単に使えるデータベースであることがご理解いただけたと思います。

Caché のユニークな機能の一つは、クラスのインスタンスとして作成されたデータが自動的にテーブルとしてもアクセスできることです。例えば、これまで定義した FS.Person が、同じ名前のテーブルとしてすでに Caché に定義されています。ここでは、それを確認してみましょう。

6.1. SQL メニュー

Caché の、SQL メニューから、定義されているデータをリレーショナルの観点から操作することができます。FS スキーマの Person テーブルのデータを SQL メニューから確認します。

① システムエクスプローラ

② SQL

③ 名前スペースを確認します。ランチャーから管理ポータルを開いた場合は%SYS 名前スペースが選択されています。

④ 名前スペース選択

⑤ スキーマのプルダウンから FS を選択

⑥ テーブルの階層を展開し、テーブル名 FS.Person を選択

⑦ 「テーブルを開く」のリンクを押下

#	ID	DOB	Gender	Name	Address_City	Address_Phone	Address_Pos
1	1	05/01/1978	M	インターシステムズ 太郎	新宿区	03-1234-5678	160-0023
2	2	10/23/1992	F	Yancik, Mark I.	Chicago	458-839-8176	I3079
3	3	11/09/1976	M	Ubertini, Janice W.	Queensbury	810-623-5629	N9697
4	4	01/25/1998	M	Townsend, Phyllis Y.	Reston	641-358-7436	Y6383
5	5	08/18/1982	M	Ihringer, Joe U.	Queensbury	835-581-2921	K9050
6	6	11/12/1974	F	Paraskiv, Barbara F.	Zanesville	702-948-3464	O1119
7	7	08/09/2010	M	Nelson, Lydia C.	Tampa	921-542-5867	T4462
8	8	12/13/1973	M	Duquesnoy, Elvis J.	Pueblo	877-464-6514	C7270
9	9	05/16/2013	F	Geoffrion, Mark G.	Queensbury	696-501-3220	T5236
10	10	12/29/1962	M	Hanson, Roberta L.	Newton	997-536-2301	J3018
11	11	01/30/1933	F	Adams, Kyra P.	Islip	275-987-2125	O2034
12	12	05/23/1938	F	Lubbar, Zeke B.	Ukiah	971-727-2742	O4593
13	13	07/06/1961	F	Frith, Lisa O.	St Louis	808-328-8861	L6662
14	14	11/05/1993	M	Vonnegut, Christine O.	Ukiah	215-209-2795	P3776

図 23 FS.Person テーブルのデータ参照

この例では 1 つのスキーマしか存在していませんが、複数のスキーマが存在すれば、それらがすべてリストされます。(名前スペースを SAMPLES に変更してから上記の操作をすると、複数のスキーマが確認できます。)

6.2. SQL 計算フィールド

テーブルをオープンした結果を見て気付いた方もいらっしゃるかもしれませんが、年齢を表す Age が表示されていません。これは、オブジェクトでは計算プロパティとして定義されているものは、自動的にテーブルで定義されないということです。

以後、テーブル定義で Age が見えるようにしていきます。

オブジェクトの定義では、AgeGet() メソッドに DOB に基づいて年齢を計算するロジックを実装しました。当然、テーブル定義からもこれを参照したいのですが、残念ながら AgeGet() メソッドにはテーブル定義からはアクセスできません。それは、AgeGet() メソッドがクラスのインスタンスを対象に実行されるインスタンスメソッドであるからです。

メソッドにはもう一種類クラスメソッドと呼ばれるものがあり、これはインスタンスなしで実行することができ、テーブル定義からもアクセスできます。まずは、AgeGet メソッドのロジックを次のような FS.Person クラスのクラスメソッドに移します。

```
ClassMethod calculateAge(dob As %Date) {  
    Quit (+$Horolog-dob)¥365  
}
```

ClassMethod 宣言で、このメソッドがクラスメソッドであることを示しています。

そして、AgeGet メソッドを次のように、先程定義したクラスメソッドを呼び出すように変更します。

```
Method AgeGet() As %Integer  
{  
    Quit ..calculateAge(..DOB)  
}
```

これで、年齢を計算するロジックの共通化ができましたので、あとは、Age をテーブルからも見えるようにするだけです。そのために、Age の定義をつぎのようにします。

```
Property Age As %Integer [ Calculated, ReadOnly,
    SqlComputeCode = { Set {Age}=##class(FS.Person).calculateAge({DOB}) },
    SqlComputed ];
```

図 24 SqlComputeCode の記述例

“SqlComputed” キーワードで SQL 計算フィールドであることを示し、その計算ロジックを “SqlComputeCode” キーワードで定義しています。ここで、{*property*} で他のプロパティ (SQL 用語ではフィールド) を参照することに注意してください。

それでは、コンパイルして先程と同様に管理ポータル の SQL 画面でテーブルをオープンしてみましょう。今度は、Age が正しく計算されて表示されていると思います。

今回は、現バージョンでは、SqlComputed コードが実装されていれば、オブジェクトアクセス用の Get メソッドを実装する必要はありません。(先ほどの AgeGet() メソッドを実装する必要はありません。)

AgeGet() メソッドがない場合の Age プロパティの定義は以下のとおりです。

```
Property Age As %Integer [ Calculated, ReadOnly,
    SqlComputeCode = { if {DOB}="" { set {Age}="" }
        else { set {Age}=( $H-{DOB} )¥365.25 } }, SqlComputed ];
```

6.3. MS-Access からのアクセス (ODBC)

Cache は ODBC からのアクセスをサポートしています。ここでは、ODBC の代表的クライアントツールである、Microsoft 社の Access (MS-Access) を使用して、ODBC 経由で Cache のデータベースにアクセスしてみます。(説明には MS-Access2007 を利用しています。)

MS-Access を起動して、データベースを作成します (既存のデータベースがあればそれを開いても結構です)。そして、「外部データ」→「ODBC データベース」→「現在のデータベースの新しいテーブルにソースデータをインポートする」を選択します。

ダイアログが表示されたら、「コンピュータデータベース」を選択します。そうすると、ODBC のデータソースを指定する画面が出てきますので、適切なデータソースを選びます。Cache では、ネームスペースごとにデータソースを定義するのが一般的ですが、Cache をインストールすると、デフォルトで USER ネームスペースに対応する CACHE User、SAMPLES ネームスペースに対応する CACHE Samples というコンピュータデータソースが作成されています。これらのデータソースの適切なものを使用するか、もしくはデータソースを必要に応じて作成してください。

データソースを選択すると、これまでの章で作成した FS.Person がテーブルとして表示されますので、それを選択して OK を押します。

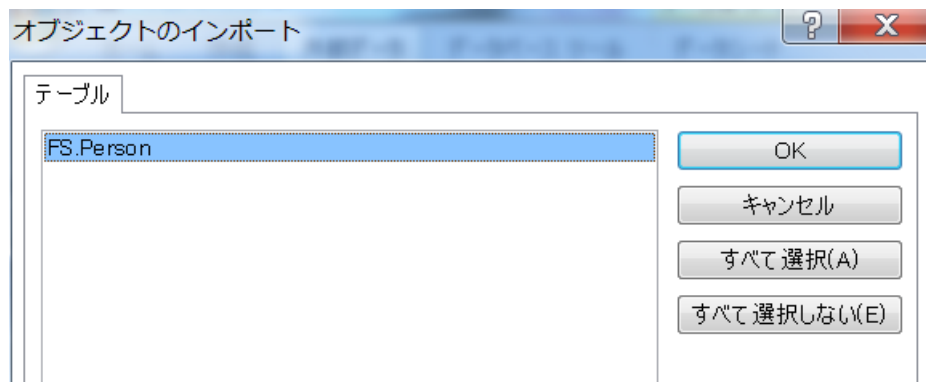


図 25 MS-Access でのテーブルインポート

そうすると、Cache の FS.Person テーブルへのリンクが生成されます。

テーブルを開くと、現在あるデータが表示されます。

すべてのテーブル		FS_Person							
テーブル1		ID	Age	DOB	Gender	Name	Address_City	Address_Pho	A
テーブル1 : テーブル		1	32	1978/05/01	M	インターシステ	新宿区	03-1234-5678	1t
FS_Person		2	49	1961/11/18	M	Zevon, Agnes G	Chicago	645-988-2329	W
FS_Person : テーブル		3	45	1965/04/11	F	DeSantis, Keith	Hialeah	508-316-9514	D
		4	23	1987/04/03	F	Djakovic, Jeff L	Boston	411-503-7677	K
		5	12	1998/11/28	M	Ravazzolo, Juan	Zanesville	653-730-7305	J
		6	71	1939/02/17	F	Marks, Emilio E	Miami	986-739-5051	Q
		7	69	1941/03/07	M	Solomon, Terry	Elmhurst	660-902-1888	J9
		8	21	1989/09/10	F	Eagleson, Bart	Bensonhurst	959-340-9249	H
		9	63	1947/10/04	M	Schultz, Liza W	Washington	868-817-7571	Q
		10	54	1956/10/17	F	Braam, Dave L	Larchmont	553-616-9931	T
		11	18	1992/10/15	F	Uhles, Marvin C	Fargo	871-556-2781	J
		12	13	1997/08/14	M	Martinez, Maria	Gansevoort	756-821-3462	S
		13	25	1985/02/19	M	Thompson, Zeld	Hialeah	575-661-6333	X
		14	17	1993/07/16	F	Black, Wilma Q	Xavier	717-460-3122	J9
		15	17	1993/05/01	F	Yancik, Lauren	Hialeah	474-714-4355	H

図 26 MS-Access でのテーブル表示 (FS.Person)

6.4. クラスクエリ

Cache では、各クラスに対してそのクラスのインスタンスを検索するためのクエリを定義し、実行することができます。このようなクエリはサーバ側で実行されます。

ここでは、名前の方前一致で FS.Person のインスタンスを検索するクエリを定義してみます。クエリは Cache スタジオで定義します。FS.Person の定義画面を開いてください。そして、「クラス」メニュー→「追加」→「クエリ...」を選択します。そうすると、ウィザード形式でクエリが定義できます。まずは、クエリの名前を定義します。ここでは、ByName としてください。次に、クエリに対する入力パラメータを定義します。今回は、検索する名前に対応するパラメータを定義します。名前 nm、タイプ %String でパラメータを一つ追加します。

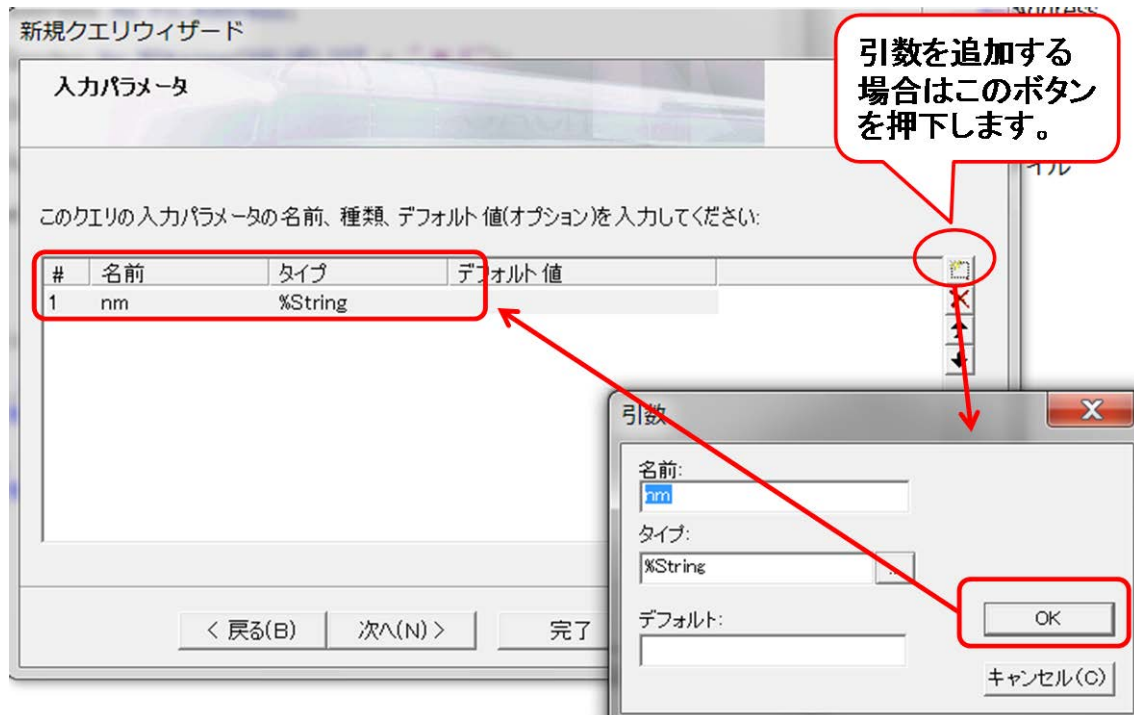


図 27 クラスクエリの追加(引数の設定)

次に、クエリの結果として取得するプロパティ(フィールド)を選択します。³

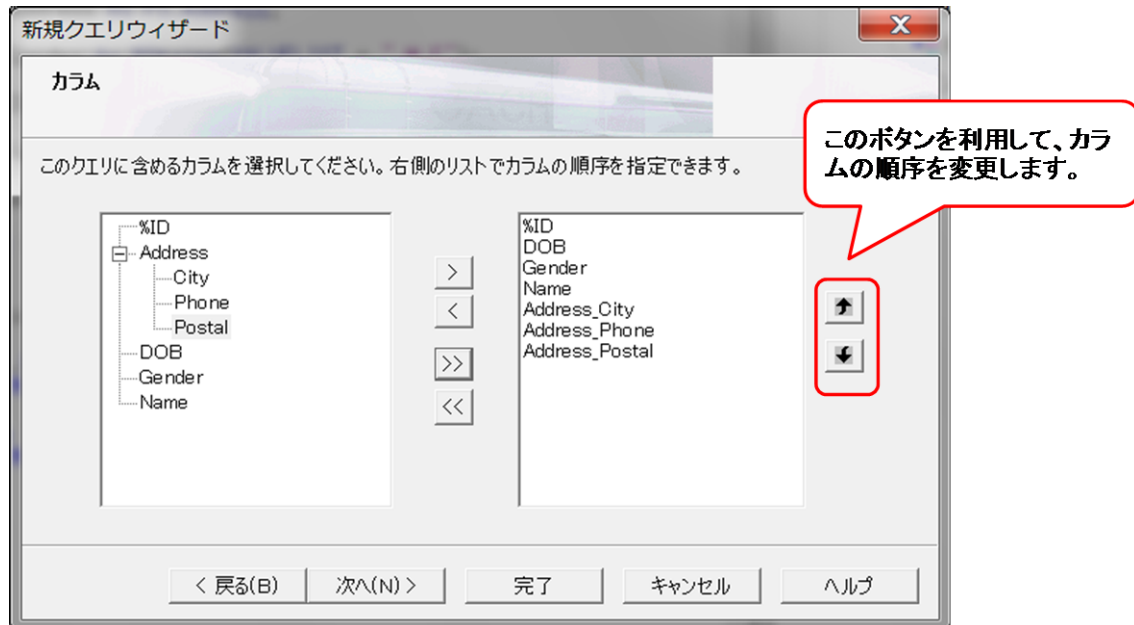


図 28 クラスクエリ 選択カラムの設定

次の画面では、条件を設定します。ここでは、Name フィールドに対する前方一致ですので、「フィールド名」Name、「条件」 %STARTSWITH、「表現」:nm を指定します。

³ 2015.1.0 のクエリウィザードでは、%ID 列を選択すると “%ID” (両端に二重引用符を付与)と表示され、このままではコンパイルが通りません。(2015.1.1 以降のバージョンでは、画面例と同じ表記に修正されます。)

ご利用中環境が、2015.1.0 でクエリウィザードで %ID 列を指定している場合は、ウィザード終了後に %ID (両端の二重引用符を取り除く)に修正してコからンパイルを行ってください。

図 29 新規クエリウィザード:条件設定の画面

次は照合順ですが、例えば、Age を指定して、年齢の順にデータを並べ替えるよう指定します。

以上でクエリの定義は終了です。

次のようなコードがコードペインに表示されます。

(2015.1 のスタジオで)

```
Query ByName(nm As %String) As %SQLQuery
{
SELECT %ID, Age, DOB, Gender, Name, Address_City, Address_Phone, Address_Postal
FROM Person
WHERE (Name %STARTSWITH :nm)
ORDER BY Age
}
```

クラスメソッドの作成には、新規メソッドウィザードを利用します。(直接コードペインで記述できますが、慣れるまでは、ウィザードを利用すると便利です。)

スタジオのメニューで、「クラス」→「追加」→「メソッド」を選択します。クラスメソッドの名前を `printByName` としています。

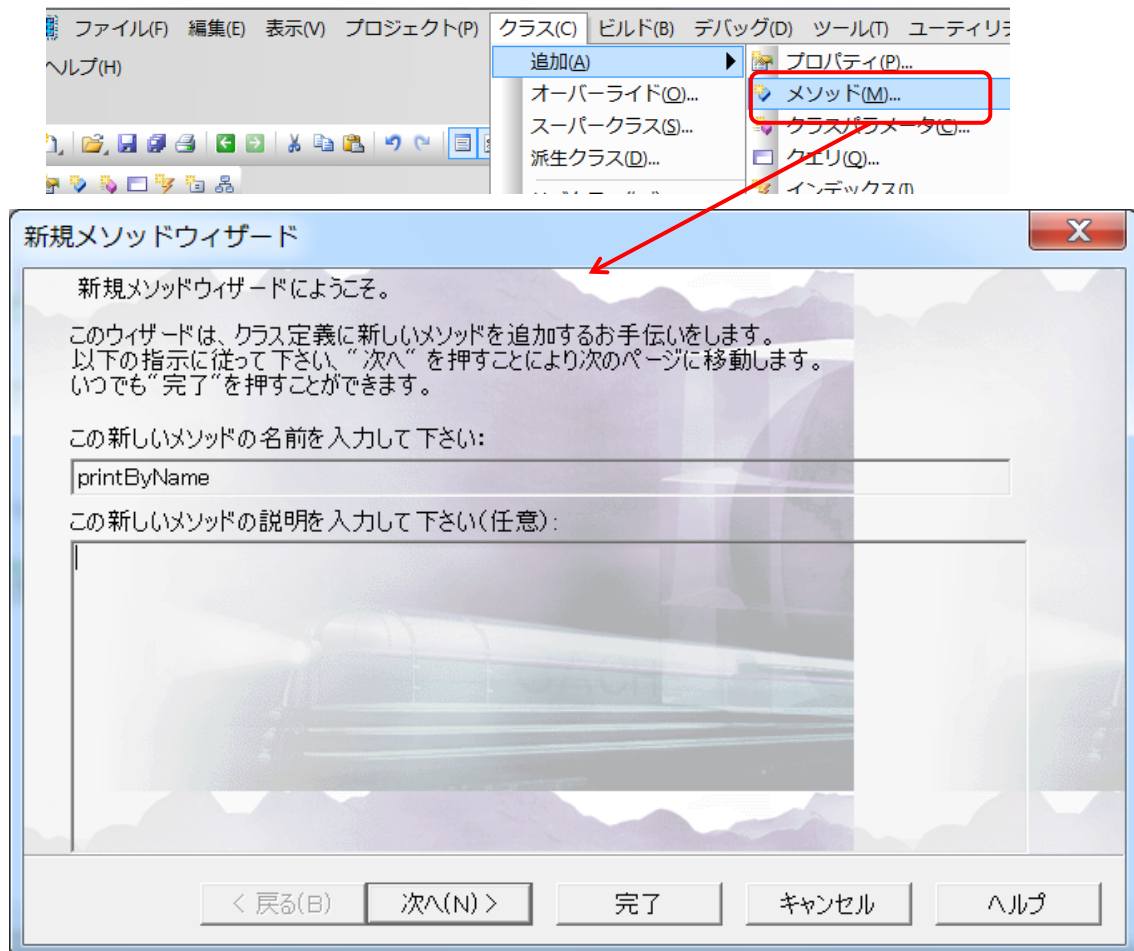


図 30 新規メソッドウィザード

`printByName()` メソッドでは、`ByName` クエリの実行を行い、検索結果をターミナル上に表示するクラスメソッドとします。引数は `ByName` クエリの実行に合わせて、名前と前方一致が行えるように、文字列の引数を 1 つ設定します。引数が与えられなかったときは、デフォルト値を利用するように、デフォルト値の設定も行います。戻り値は設定なしとします。

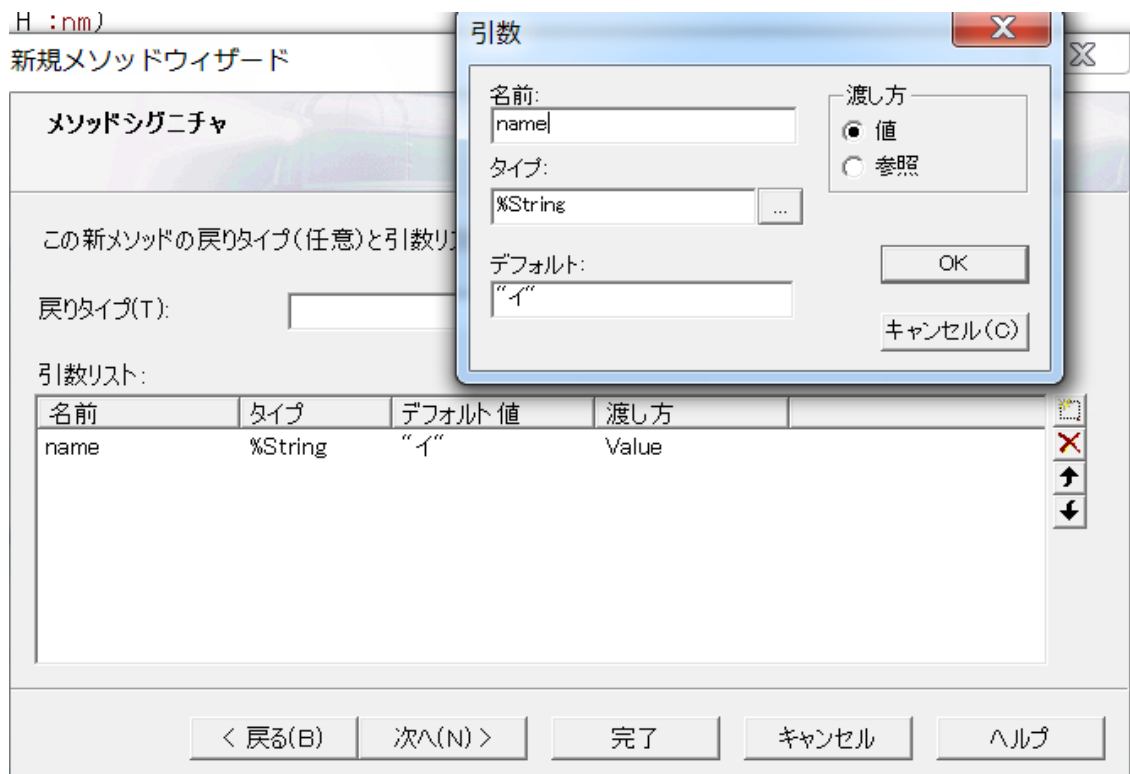


図 31 新規メソッドウィザード 引数の指定

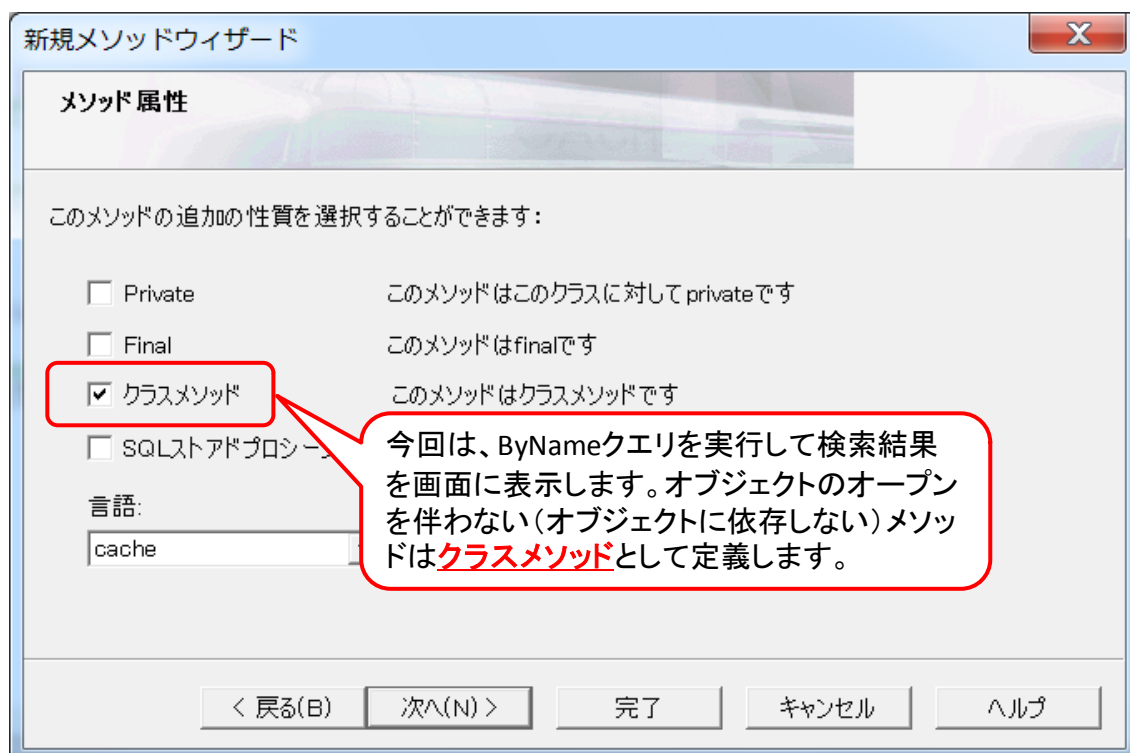


図 32 新規メソッドウィザード クラスメソッドの設定

新規メソッドウィザードの最終画面でコードを記述します。コードは、先頭文字にタブを 1 つ挿入してから、コマンドを記述します。（先頭文字から記述した文字は、全てラベル名と判断されます。）コードは、一旦ウィザードを「完了」ボタンで終了し、コードペインで記述することもできます。

（コード例の // から始まる行はコメント行です。）

```

43 ClassMethod printByName(name As %String = "イ")
44 {
45     // %New("パッケージ名.クラス名:クエリ名") で実行するクエリを指定
46     set rset=##class(%ResultSet).%New("FS.Person:ByName")
47
48     /* ClassName、QueryName プロパティを利用して指定する方法もあります。
49     set rset.ClassName="FS.Person"
50     set rset.QueryName="ByName"
51     */
52
53     // SELECTに%Dateのタイプを持ったDOBを指定しています。
54     // RuntimeMode プロパティを利用して、表示変換を行います。（サーバー側のみ）
55     set rset.RuntimeMode=1 // 1:ODBC形式
56
57     // クエリ時の実行 クエリの引数はExecute()の引数に指定します
58     do rset.Execute(name)
59
60     // 結果セットの行を移動します。最終行に到達すると0を返します
61     while (rset.Next()'=0) {
62
63         // Get("列名") でカラムの値を取得します
64         write !,rset.%Get("Name"), " - ",rset.%Get("Address_City")
65         write " - ",rset.%Get("DOB")," - ",rset.%Get("Gender")
66
67         /*
68         // <<補足>>GetData(カラム番号) でも取得できます。（1から開始）
69         // GetColumnCount() から結果セット中のカラム数を取得できます
70         write !
71         for i=1:1:rset.GetColumnCount() {
72             write rset.GetData(i)," - "
73         }
74         */
75     }
76
77     // 結果セットの終了
78     do rset.Close()
79     kill rset
80     quit
81 }

```

図 33 printByName()のコード:ByName クエリの実行

メソッドのコードでは、クエリの実行に%Library.ResultSet クラスのインスタンスを作成しています。（%Library パッケージは省略できますので、例では、%ResultSet と記述しています。）%ResultSet のインスタンス作成後、クラスクエリを実行するため、Execute()メソッドを実行しています。クエリに引数がある場合は、Execute()の引数に指定します。

続いて、検索結果からデータを取得／表示するため Next()メソッドで行を移動します。Next()メソッドの戻り値が 0 になるとき、最終行に到達したこととなるので 0 が来るまでループします。カラムの値取得には、Get()メソッドの引数にカラム名を文字列として指定しています。

では、作成したクラスメソッドを実行します。まずは、クラス定義をコンパイルしてください。Caché ターミナル上で、次のようなコマンドを実行します。

```
do ##class(FS.Person).printByName("A")
```

そうすると、Name フィールドが“A”で始まるレコードが表示されるはずです。色々なパラメータで実行してみてください。

《補足》 ##class(パッケージ名.クラス名).クラスメソッド名(引数) の順序で指定します。

6.5. インデックスの生成

前のセクションでは、名前の方前一致で FS.Person を検索するクエリを作成しました。数千件程度のレコードから探す場合は問題になりませんが、通常大量のレコードから検索する場合は、インデックスを作成して検索時間の短縮を図るのが普通です。

Cache は、2 つの種類のインデックスをサポートしています。一つは通常のインデックス、もう一つはビットマップインデックスです。ビットマップインデックスは、比較的値の散らばりが小さいフィールドに対して、複雑な条件で検索を行うときに有用なインデックスです。ビットマップインデックスには、インデックスが占めるディスク領域が少なくすむという特性もあります。

ここでは、インデックスの生成の仕方の例として、通常インデックスを定義してみます。Cache スタジオで、FS.Person の定義画面を開きます。そして、FS.Person の定義の中に、次のような行を追加します。

```
Index NameIdx on Name;
```

この定義は、NameIdx という名前のインデックスを、Name フィールドに対して定義しています。

この段階では、インデックスは定義されたのみです。したがってこれ以降に追加されるレコードにはインデックスが自動的に生成されますが、既存のデータに対しては明示的にインデックスを生成する必要があります。このようなインデックスの再構築は、SQL メニューからも可能ですが、Cache ターミナルで次のようなコマンドを実行することもできます（数千件程度なら構築は一瞬で終わります）。

```
Do ##class(FS.Person).%BuildIndices()
```

これで、既存のデータに対してもインデックスが構築されました。数千件程度のデータであればそれ程検索速度には差は出ないでしょうが、大量データに対しては大きな効果が現れます。

ここまでの作業のサンプルは、Step3 フォルダにあります。

7. 最後に

以上、Caché を初めて体験する方を対象に、Caché の持つ基本的機能についてご紹介してきました。ここまでお読みいただいた方は、すでに Caché を使ったアプリケーションを構築するのに必要な基礎知識を習得されたことになります。

もちろん、Caché にはここにご紹介し切れなかった機能がたくさんあります。Java や .NET との連携、XML、Web アプリケーションの構築、Web サービス、複雑なオブジェクト連携などです。

これらの機能については、マニュアルや、今後弊社から出される資料をご参照ください。またFAQ サイトも合わせてご参照ください。(<http://faq.intersystems.co.jp/>)

さらに、より詳しいことがお聞きになりたい場合は、インターシステムズジャパン(株)までお問い合わせください。

インターシステムズジャパン(株)

<http://www.intersystems.co.jp/>

TEL: 03-5321-6200(代表)

Email: jpinfo@intersystems.com