



# About the Caché Populate Utility

Version 2007.1

04 June 2007

*About the Caché Populate Utility*

Caché Version 2007.1 04 June 2007

Copyright © 2007 InterSystems Corporation

All rights reserved.

This book was assembled and formatted in Adobe Page Description Format (PDF) using tools and information from the following sources: Sun Microsystems, RenderX, Inc., Adobe Systems, and the World Wide Web Consortium at [www.w3c.org](http://www.w3c.org). The primary document development tools were special-purpose XML-processing applications built by InterSystems using Caché and Java.



Caché WEBLINK, Distributed Cache Protocol, M/SQL, N/NET, and M/PACT are registered trademarks of InterSystems Corporation.



InterSystems Jalapeño Technology, Enterprise Cache Protocol, ECP, and InterSystems Zen are trademarks of InterSystems Corporation.

All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

This document contains trade secret and confidential information which is the property of InterSystems Corporation, One Memorial Drive, Cambridge, MA 02142, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

For Support questions about any InterSystems products, contact:

**InterSystems Worldwide Customer Support**

Tel: +1 617 621-0700

Fax: +1 617 374-9391

Email: [support@InterSystems.com](mailto:support@InterSystems.com)

# Table of Contents

<b>About the Caché Populate Utility</b> .....	<b>1</b>
1 Simple Use .....	1
2 Embedded Serial Classes .....	2
3 Advanced Use .....	3
4 The OnPopulate() Method .....	4
5 Details .....	4
6 Quick Reference to %PopulateUtils Methods .....	5



# About the Caché Populate Utility

Once you have defined your first few classes, you probably want to start testing them. To do that, you need to put some data into the database. This can be done using Caché ObjectScript commands, such as:

```
Set obj=##class(Person).%New()  
Set obj.Name="Kerouac,Jack"  
Set obj.DOB="+$h  
Write obj.%Save()  
Do obj.%Close()
```

But this soon becomes boring and slow for most developers. As development proceeds, you may need to build a series of methods (or a routine) that inserts valid and meaningful data into the database, or perhaps even loads some real data itself.

Before doing this, you may want to just load some almost-random data into the database, so that you can exercise your classes and the User Interface. This is where the %Populate class can help.

## 1 Simple Use

Using the %Populate class can be very simple, or you can go to a lot of trouble to 'customize' the data that it creates, to make it look more realistic and complete. In the simplest case, you simply add %Populate as an additional superclass of your new class, such as:

```
super = %Persistent, %Populate;
```

Once your class has been recompiled, you will have access to a new method, **Populate()**, which can be called as follows:

```
Do ##class(Person).Populate()
```

This will create 10 instances of your class and store them within the database, which you can see by running an SQL query against your object or by using a utility (such as the Caché System Management Portal) on the global `oo<class>D`, in this case `ooPersonD` (and `ooPersonI`, if you have defined any indices).

The **Populate()** method takes, as an argument, an integer specifying how many instances to create, and returns a value showing how many were saved successfully (you may have defined some constraints on your class that are not satisfied by all randomly-generated data). In the following case:

```
Write ##class(Person).Populate(20)
19
```

The **Populate()** method tried to insert another 20 instances to the class, but one insert operation failed.

Note that you must manually delete the class extent (set of all object instances) if you want to start with an empty one. You can do this using the **%DeleteExtent()** method of the persistent interface (see [The Persistence Interface](#)).

The current version of %Populate will assign random values to most 'simple' attributes and references (as long as the referenced class has some instances), but will not give values to collections, etc. (see below).

Thus an attribute defined as type %Integer will be assigned a random integer (obeying the values of the MINVAL and MAXVAL parameters), while a %Date attribute will get a random date.

A %String will be assigned a random string value (see below).

## 2 Embedded Serial Classes

If your class contains an embedded serial class, such as a Home attribute within a Person class, which might be an instance of an Address class, then by default this will not be given a value by %Populate.

You can cause random values to be assigned to embedded classes by defining the POPSPEC parameter for both the embedded and the host class. The POPSPEC parameter tells %Populate which method to use when populating a particular attribute.

So, for example, if the Address class contains the attributes City and State, you might define:

```
parameter POPSPEC { default = "City:City(),State:USState()"; }
```

This specifies that for the City attribute, you want to use the **City()** method in the class %PopulateUtils, and for the State attribute you want to use the **USState()** method in %PopulateUtils.

The class %PopulateUtils provides a library of commonly-used data generation methods.

Then, within your main class (Person) you would define POPSPEC as:

```
parameter POPSPEC { default="Home:.Home.PopulateSerial()"; }
```

which specifies that for the Home attribute, you want to use the **PopulateSerial()** method of the Home attribute.

## 3 Advanced Use

Sometimes, you want more control over the data that is generated, and you can achieve this by using the `POPSPEC` parameter introduced above and a method selected from the `%PopulateUtils` library or written yourself.

The `POPSPEC` parameter uses the following syntax :

```
Parameter POPSPEC { default="<att>:<spec>,<att>:<spec>,. . . "; }
```

Where:

- `att` is the attribute name in the class
- `spec` is the data generation specification, and can be in one of the following formats :
  - `<method>` references a method in `%PopulateUtils`
  - `.<method>` references a method of the current instance
  - `##class(<class>).<method>` references a method directly

Any attributes not listed in `POPSPEC` will get default data generated as described above.

For example if `POPSPEC` is specified as:

```
Parameter POPSPEC { default="PName:Name("F"),
  Code:..MakeCode(), SSN:##class(Utils).MakeSSN()"; }
```

Then attribute `PName` will be populated by calling the method `##class(%PopulateUtils).Name("F")`.

The attribute `Code` will be valued using `obj.MakeCode()` where `obj` is the handle to the object being created, so this expects you to code the method **MakeCode()** into your class.

The attribute `SSN` will be populated using `##class(Utils).MakeSSN()`.

The attribute `Number` (defined as type `%Integer`) will receive a random integer.

You can achieve the same level of control by defining the `POPSPEC` parameter on a per-property basis. The usage is the same as described above but only specifies the behavior of a single property:

```
attribute Name { type = %String(MAXLEN=50, POPSPEC=.FirstName()); }
```

# 4 The OnPopulate() Method

For additional control over the generated data you can define an **OnPopulate()** method within your class. If an **OnPopulate()** method is defined then the **Populate()** method will call it for each object it generates. The method is called after assigning values to the properties but before the object is saved to disk. When you call the **Populate()** method it, in turn, checks for the existence of the **OnPopulate()** method and calls it for each object it generates. The method is called after assigning values to the properties but before they are saved.

For example, if you have a stream attribute, Memo, and you would like to assign a value to when populating, you can provide an **OnPopulate()** method:

```
method OnPopulate()
  returntype = %Status;
  code = {
    Do ..Memo.Write("Default value");
    QUIT $$$OK
  }
}
```

# 5 Details

This section describes in some detail how %Populate works, so that you can understand what it does, and how to influence it

The %Populate class contains two method generators **Populate()** and **PopulateSerial()**, which are quite similar. Each persistent or serial class inheriting from %Populate will get one or the other of these two methods included.

We will describe only the **Populate()** method here.

The **Populate()** method is a loop, repeated for the requested number of objects.

Inside the loop, the code

- Creates a new object
- Values the attributes
- Saves and Closes the object

A simple attribute with no overriding POPSPEC parameter is populated using code similar to:

---

```
Set obj.Description=##class(%PopulateUtils).String(50)
```

Using a library method from %PopulateUtils via a "Name:Name()" specification would generate:

```
Set obj.Name=##class(%PopulateUtils).Name()
```

An embedded Home attribute might create code like:

```
Do obj.HomeSetObject(obj.Home.PopulateSerial())
```

The generator loops through all the properties of the class, and creates code for some of the properties, as follows:

- if the property is private, quit
- else if the property is calculated, quit
- else if the property is a collection, quit
- else if the property has an initial expression, quit
- else if there is a POPSPEC override for the property, use that, quit
- else if the property is a reference, (first time: build a list of random id's), take one from the list of random id's, quit
- else if the property name is one of "Name", "SSN", "Company", "Title", "Phone", "Zip" then use the corresponding library method "Name", "SSN", "Company", "Title", "USPhone", "USZip" and quit
- else if the property type is %String, %Integer, %Date, %Name then use the corresponding library method "String(MAXLEN)", "Integer(MINVAL,MAXVAL)", "Date", "Name" and quit
- else set the property to ""

## 6 Quick Reference to %PopulateUtils Methods

This section provides a brief description of available %PopulateUtils methods. For a complete and up-to-date listing, refer to the %PopulateUtils class documentation.

Valid POPSPEC values are:

- `City()` — City name string.
- `Company()` — Company name string.
- `Currency()` or `Currency(min, max)` — Random number string in format 4n.4n.
- `Date()` or `Date(start, end)` — Date string in format nnnnn.

## Quick Reference to %PopulateUtils Methods

---

- `FirstName()` or `FirstName(n)` — First name string. Generates only male names if  $n = 1$ , only female names if  $n = 2$ .
- `LastName()` — Last name string.
- `Mission()` — Phrase describing company's mission.
- `Name()` or `Name(n)` — Full name string in format `last, m. first`. Generates only male names if  $n = 1$ , only female names if  $n = 2$ .
- `SSN()` — Social Security Number string in format `nnn-nn-nnnn`.
- `Street()` — Street address string.
- `String()` or `String(len)` — String of random characters.
- `TimeStamp()` — Timestamp in format `yyyy-mm-dd hh:mm:ss`.
- `Title()` — Phrase describing position in company.
- `USPhone()` — US phone number.
- `USState()` — US state abbreviation.
- `USZip()` — US five-digit zip code.
- `ValueList(",value1,valueN")` — Quoted, comma delimited list of values.
- `VarString()` or `VarString(maxLen)` — Variable length string of random characters.