



FileMan to Caché SQL Mapping Tool

Version 1.0 build 8

Developer Manual

June 15, 2006

1. Introduction.....	4
1.1 Overview	4
1.2 System Requirements.....	4
2. Installation.....	5
2.1 KIDS build (CASH1_0.KID)	5
Use XPD Main to install the file CASH1_0.KID.	5
2.2 Manual Installation.....	9
1) Install the routines in file CASH1_0.RO using %RI.	9
2) Install the Globals in file CASH1_0.GOGEN using %GIGEN.	9
3) Run the Initialization script CASHI.....	10
3. Using the Tool.....	13
3.1 START^CASH	13
Format.....	13
Parameters	13
Return Value	16
Examples.....	16
3.2 CREATE^CASH.....	19
Format.....	20
Parameters	20
Return Value	22
Examples.....	23
3.3 ALL^CASH	25
Format.....	25
Parameters	25
Examples.....	26
3.4 Editing Classes.....	27
Expanded Pointer Fields ("E" flag)	27
Output Transforms	27
4. Caveats and Warnings.....	29
4.1 System Performance.....	29
4.2 Security Considerations.....	29
4.3 Database Size and Journaling.....	29
4.4 Re-Installation of Custom Datatypes (4.1.*)	29
4.5 Running ALL^CASH on Caché 4.1.*	30

4.6 Very Large FileMan Files (250+ Fields)	30
4.6.1 Maximum Fields Per Class (250)	30
4.6.2 Maximum Multiples per Class (75)	31
4.6.3 Caché Version 5.1	31
4.6.4 CASH Parameter File (#15050.14)	31
Appendix A	32
The Caché 4.1.* class compiler	32
The size of ^oddDEF affects compilation speed	32
ALL ^CASH	32
"Memory washout"	32
Workarounds	33

1. Introduction

1.1 Overview

This FileMan to Cache SQL Mapping Tool has been designed to allow developers to quickly and easily create custom Cache SQL mappings for their FileMan Files. The developer can choose to map just one, many or all FileMan files in a namespace, with a variety of options to serve many different purposes.

It has been designed to be lightweight and flexible and to have no dependencies outside of Caché and CASH.FileMan. Where possible the tool has been developed as a standard FileMan patch, so it can be released as a normal KIDS build.

For each file, the developer can control whether:

- All the fields are mapped or just a selected few.
- Multiples are expanded as Sub-Classes (Child Tables) or ignored.
- Pointers are created as References (Foreign Keys) to other Classes (which will be created if necessary), or expanded as Computed Fields within the same Class (the .01 field is used by default, but can be changed), or both!
- Output Transforms are used or not, on a field by field basis.

There are also several advanced options available for Cache 5.0.* users:

- The %XML.Adaptor Super-Class can be added to the Class, with a generated XMLDump() method that will export the whole file to XML. The MAXROWS parameter can be used to limit this for very large files!
- The %CSP.Page Super-Class can be added to the Class, with a generated OnPage() method that calls the XMLDump() method.
- A sister Class can be created with the %SOAP.WebService Super-Class, which contains generated SQL Stored Procedures for each index in the Class, all of which are also exposed as WebMethods.

1.2 System Requirements

You must be running a standard Caché installation, Version 4.1.*, 5.0.*, or higher.

The Maximum Memory Per Process (KB) setting should be at least 1024 (1MB) for this tool.

The minimum recommended requirement for Caché SQL is at least 2048 (2MB), but I have found 1MB to be enough for most purposes. If you plan to use Web Services in Caché 5.0+, then you will almost certainly need 2MB.

You need a working FileMan installation. That's it!

2. Installation

The easiest way to install this tool is to use the KIDS build. The installation should take no more than 2 minutes (depending on your machine and Version of Cache). You can also install manually from the Routine Output (.RO) and Partial Global Output (.GOGEN) files.

Note: If you are re-installing or updating on Caché Version 4.1.* and you have a large number of Classes (thousands) that have already been built using this tool, the CASH.FileMan.* Datatypes can take a long time to compile! This is probably because Caché is building a list of dependencies. See Appendix A for more details.

2.1 KIDS build (CASH1_0.KID)

Use XPD Main to install the file CASH1_0.KID.

I have highlighted my User Interaction in the following transcript (with comments added outside of the boxes):

```
D ^XUP

Setting up programmer environment
This is a TEST account.

Terminal Type set to: C-VT320

Select OPTION NAME: XPD MAIN      Kernel Installation & Distribution System

    Edits and Distribution ...
    Utilities ...
    Installation ...
    Patch Monitor Main Menu ...

Select Kernel Installation & Distribution System Option: Installation

    1      Load a Distribution
    2      Verify Checksums in Transport Global
    3      Print Transport Global
    4      Compare Transport Global to Current System
    5      Backup a Transport Global
    6      Install Package(s)
          Restart Install of Package(s)
          Unload a Distribution

Select Installation Option: 1 Load a Distribution
Enter a Host File: CASH1_0.KID

KIDS Distribution saved on May 15, 2006@22:50:46
Comment: FM To Cache SQL Mapping Tool - Version 1.0 Build 8

This Distribution contains Transport Globals for the following Package(s):
    FM TO CACHE SQL 1.0
Distribution OK!

Want to Continue with Load? YES// <Enter>
Loading Distribution...

    FM TO CACHE SQL 1.0
```

Use INSTALL NAME: FM TO CACHE SQL 1.0 to install this Distribution.

- 1 Load a Distribution
- 2 Verify Checksums in Transport Global
- 3 Print Transport Global
- 4 Compare Transport Global to Current System
- 5 Backup a Transport Global
- 6 Install Package(s)
Restart Install of Package(s)
Unload a Distribution

Select Installation Option: **6** Install Package(s)

Select INSTALL NAME: **FM TO CACHE SQL 1.0** Loaded from Distribution

Loaded from Distribution 5/18/06@15:36:57

=> FM To Cache SQL Mapping Tool - Version 1.0 Build 8 ;Created on May 15

This Distribution was loaded on May 18, 2006@15:36:57 with header of
FM To Cache SQL Mapping Tool - Version 1.0 Build 8 ;Created on May 15,
2006@22:50:46

It consisted of the following Install(s):

FM TO CACHE SQL 1.0

Checking Install for Package FM TO CACHE SQL 1.0

Install Questions for FM TO CACHE SQL 1.0

Incoming Files:

- 15050.11 CASH FM CLASS MAP
- 15050.12 CASH CUSTOM DATATYPES
- 15050.13 CASH ERRORS
- 15050.14 CASH PARAMETER FILE (including data)
- 15050.19 CASH SQL RESERVED WORDS

Want KIDS to INHIBIT LOGONS during the install? YES// **NO**

Want to DISABLE Scheduled Options, Menu Options, and Protocols? YES// **NO**

Enter the Device you want to print the Install messages.

You can queue the install by enter a 'Q' at the device prompt.

Enter a '^' to abort the install.

DEVICE: HOME// **<Enter>** CONSOLE

Complete FM TO CACHE SQL 1.0

Install Started for FM TO CACHE SQL 1.0 :

May 18, 2006@15:37:09

Build Distribution Date: May 15, 2006

Installing Routines: 100%

May 18, 2006@15:37:09

Running Pre-Install Routine: CLEAN^CASHI

Installing Data Dictionaries: 100%

May 18, 2006@15:37:09

Installing Data:

May 18, 2006@15:37:09

Running Post-Install Routine: POST^CASHI

Initializing SQL Reserved Words File (#15050.19), if necessary.

%AFTERHAVING added

%ALPHAUP added

Note: 269 total terms will be added. Each term added to file 15050.19 will be listed for the user, but they have not been listed here for space considerations.

WRITE added

YEAR added

Removing old Error Log data (#15050.13)

Create and compile the CASH Utility Classes.

Deleting class CASH.FMField

done.

Deleting class CASH.FMFile

done.

Compilation started on 05/18/2006 15:37:10

Compiling class CASH.FMFile

Compiling class CASH.FMField

Compiling table CASH.FMField ...

Compiling table CASH.FMFile ...

Compiling routine CASH.FMFile.1

Compiling routine CASH.FMField.1

Compilation finished successfully.

Deleting class CASH.Utility

done.

Compilation started on 05/18/2006 15:37:11

Compiling class CASH.Utility

Compiling routine CASH.Utility.1

Compilation finished successfully.

Create and compile the latest CASH.FileMan Datatypes.

Deleting class CASH.FileMan.Date

done.

Compilation started on 05/18/2006 15:37:11

Compiling class CASH.FileMan.Date

Compiling routine: CASH.FileMan.Date.G1.MAC.....

Compiling routine CASH.FileMan.Date.1

Compilation finished successfully.

Deleting class CASH.FileMan.DateTime

done.

Compilation started on 05/18/2006 15:37:11

Compiling class CASH.FileMan.DateTime

Compiling routine: CASH.FileMan.DateTime.G1.MAC.....

Compiling routine CASH.FileMan.DateTime.1

Compilation finished successfully.

Deleting class CASH.FileMan.Numeric

done.

Compilation started on 05/18/2006 15:37:11

Compiling class CASH.FileMan.Numeric

Compiling routine: CASH.FileMan.Numeric.G1.MAC.....

Compiling routine CASH.FileMan.Numeric.1

Compilation finished successfully.

Deleting class CASH.FileMan.Pointer

done.

Compilation started on 05/18/2006 15:37:11

Compiling class CASH.FileMan.Pointer

Compiling routine: CASH.FileMan.Pointer.G1.MAC.....

Compiling routine CASH.FileMan.Pointer.1

Compilation finished successfully.

```
Deleting class CASH.FileMan.SetOfCodes done.
Compilation started on 05/18/2006 15:37:11
Compiling class CASH.FileMan.SetOfCodes .....
Compiling routine: CASH.FileMan.SetOfCodes.G1.MAC.....
Compiling routine CASH.FileMan.SetOfCodes.1
Compilation finished successfully.

Deleting class CASH.FileMan.String done.
Compilation started on 05/18/2006 15:37:11
Compiling class CASH.FileMan.String .....
Compiling routine: CASH.FileMan.String.G1.MAC.....
Compiling routine CASH.FileMan.String.1
Compilation finished successfully.

Deleting class CASH.FileMan.StringDateTime done.
Compilation started on 05/18/2006 15:37:11
Compiling class CASH.FileMan.StringDateTime .....
Compiling routine: CASH.FileMan.StringDateTime.G1.MAC.....
Compiling routine CASH.FileMan.StringDateTime.1
Compilation finished successfully.

Deleting class CASH.FileMan.VariablePointer done.
Compilation started on 05/18/2006 15:37:11
Compiling class CASH.FileMan.VariablePointer .....
Compiling routine: CASH.FileMan.VariablePointer.G1.MAC.....
Compiling routine CASH.FileMan.VariablePointer.1
Compilation finished successfully.

Updating Routine file... 100%
Updating KIDS files... 100%
FM TO CACHE SQL 1.0 Installed.
    May 18, 2006@15:37:11
Install Message sent #3021
Install Completed
```

You are now ready to use the Mapping Tool!

Note: If for some reason the Custom Datatypes were not created and/or the SQL Reserved Word File (#15050.19) were not created you can run or re-run ^CASHI at any time.

2.2 Manual Installation

It is recommended that you use the KIDS install, which is a one step process.

Doing a manual install is a three step process, detailed below:

1) Install the routines in file CASH1_0.RO using %RI.

I have highlighted my User Interaction in the following transcript:

```
D ^%RI

Input routines from Sequential
Device: CASH1_0.RO Parameters: "RS"=>

File written by Cache for Windows NT using %RO on 15 May 2006 10:50 PM
with extension INT and with description:
FM To Cache SQL Mapping Tool - Version 1.0 Build 8

( All Select Enter List Quit )

Routine Input Option: All Routines

If a selected routine has the same name as one already on file,
shall it replace the one on file? No => Yes
Recompile? Yes => Yes
Display Syntax Errors? Yes => Yes

^ indicates routines which will replace those now on file.
@ indicates routines which have been [re]compiled.
- indicates routines which have not been filed.

CASH.INT^@      CASH0.INT^@      CASH1.INT^@      CASH2.INT^@      CASH3.INT^@
CASHC.INT^@     CASHC0.INT^@     CASHC1.INT^@     CASHC2.INT^@     CASHC3.INT^@
CASHC4.INT^@     CASHC5.INT^@     CASHCN.INT^@     CASHCU.INT^@     CASHD.INT^@
CASHDT01.INT^@  CASHDT02.INT^@  CASHDT03.INT^@  CASHDT04.INT^@  CASHDT05.INT^@
CASHDT06.INT^@  CASHDT07.INT^@  CASHDT08.INT^@  CASHF.INT^@      CASHFN11.INT^@
CASHFN12.INT^@  CASHFN13.INT^@  CASHFN14.INT^@  CASHFN19.INT^@  CASHI.INT^@
CASHN.INT^@     CASHR.INT^@     CASHR0.INT^@     CASHU.INT^@     CASHUT.INT^@
CASHUT01.INT^@  CASHUT02.INT^@  CASHV4C.INT^@    CASHV4C0.INT^@  CASHV4C1.INT^@
CASHV4C2.INT^@  CASHV4C3.INT^@  CASHV4C4.INT^@  CASHV4D.INT^@    CASHV4UT.INT^@

45 routines processed.
```

2) Install the Globals in file CASH1_0.GOGEN using %GIGEN.

(You can't use %GI as this export contains partial global references).

I have highlighted my User Interaction in the following transcript:

```
D ^%GIGEN

Device: CASH1_0.GOGEN Parameters: "R"=>
Transfer entire set of files? Yes
Transferring files on May 15 2006 at 10:55 PM
From global ^CASH(15050.11,0
To global ^CASH(15050.11,0

OK to transfer? Yes
```

```
Transfer completed
Transferring files on May 15 2006 at 10:55 PM
From global ^CASH(15050.12,0
To global ^CASH(15050.12,0
```

OK to transfer? **Yes**

```
Transfer completed
```

Note: There are about 36 partial global references, only the first and last few are listed here. In Caché 5.0 you may need to confirm each reference!

```
Transferring files on May 15 2006 at 10:55 PM
From global ^DD(15050.141
To global ^DD(15050.141
```

OK to transfer? **Yes**

```
Transfer completed
Transferring files on May 15 2006 at 10:55 PM
From global ^DD(15050.19
To global ^DD(15050.19
```

OK to transfer? **Yes**

```
Transfer completed
Done for this set of files.
```

3) Run the Initialization script CASHI.

I have highlighted my User Interaction in the following transcript (with comments added outside of the boxes):

```
D POST^CASHI
```

```
Initializing SQL Reserved Words File (#15050.19), if necessary.
```

```
%AFTERHAVING added
%ALPHAUP added
```

Note: 269 total terms will be added. Each term added to file 15050.19 will be listed for the user, but they have not been listed here for space considerations.

```
WRITE added
YEAR added
```

```
Removing old Error Log data (#15050.13)
```

```
Create and compile the CASH Utility Classes.
```

```
Deleting class CASH.FMField done.
Deleting class CASH.FMFile done.
Compilation started on 05/18/2006 15:19:05
Compiling class CASH.FMFile .....
Compiling class CASH.FMField .....
Compiling table CASH.FMField ...
Compiling table CASH.FMFile ...
Compiling routine CASH.FMFile.1
Compiling routine CASH.FMField.1
Compilation finished successfully.
```

```
Deleting class CASH.Utility done.
Compilation started on 05/18/2006 15:19:06
```

```
Compiling class CASH.Utility .....
Compiling routine CASH.Utility.1
Compilation finished successfully.

Create and compile the latest CASH.FileMan Datatypes.

Deleting class CASH.FileMan.Date ..... done.
Compilation started on 05/18/2006 15:19:06
Compiling class CASH.FileMan.Date .....
Compiling routine: CASH.FileMan.Date.G1.MAC.....
Compiling routine CASH.FileMan.Date.1
Compilation finished successfully.

Deleting class CASH.FileMan.DateTime ..... done.
Compilation started on 05/18/2006 15:19:06
Compiling class CASH.FileMan.DateTime .....
Compiling routine: CASH.FileMan.DateTime.G1.MAC.....
Compiling routine CASH.FileMan.DateTime.1
Compilation finished successfully.

Deleting class CASH.FileMan.Numeric ..... done.
Compilation started on 05/18/2006 15:19:06
Compiling class CASH.FileMan.Numeric .....
Compiling routine: CASH.FileMan.Numeric.G1.MAC.....
Compiling routine CASH.FileMan.Numeric.1
Compilation finished successfully.

Deleting class CASH.FileMan.Pointer ..... done.
Compilation started on 05/18/2006 15:19:06
Compiling class CASH.FileMan.Pointer .....
Compiling routine: CASH.FileMan.Pointer.G1.MAC.....
Compiling routine CASH.FileMan.Pointer.1
Compilation finished successfully.

Deleting class CASH.FileMan.SetOfCodes ..... done.
Compilation started on 05/18/2006 15:19:06
Compiling class CASH.FileMan.SetOfCodes .....
Compiling routine: CASH.FileMan.SetOfCodes.G1.MAC.....
Compiling routine CASH.FileMan.SetOfCodes.1
Compilation finished successfully.

Deleting class CASH.FileMan.String ..... done.
Compilation started on 05/18/2006 15:19:06
Compiling class CASH.FileMan.String .....
Compiling routine: CASH.FileMan.String.G1.MAC.....
Compiling routine CASH.FileMan.String.1
Compilation finished successfully.

Deleting class CASH.FileMan.StringDateTime ..... done.
Compilation started on 05/18/2006 15:19:06
Compiling class CASH.FileMan.StringDateTime .....
Compiling routine: CASH.FileMan.StringDateTime.G1.MAC.....
Compiling routine CASH.FileMan.StringDateTime.1
Compilation finished successfully.

Deleting class CASH.FileMan.VariablePointer ..... done.
Compilation started on 05/18/2006 15:19:06
Compiling class CASH.FileMan.VariablePointer .....
Compiling routine: CASH.FileMan.VariablePointer.G1.MAC.....
Compiling routine CASH.FileMan.VariablePointer.1
Compilation finished successfully.
```

Note: If the Datatypes already existed, they are deleted before being re-created and compiled.

You are now ready to use the Mapping Tool!

Final Draft

3. Using the Tool

3.1 START^CASH

This is the main call of the FM to Caché SQL Mapping Tool, and may be the only one you need!

It initiates the Discovery process that loops through the ^DIC and ^DD globals for the FileMan File(s), creating a template for each (in File #15050.11), that is used to create and compile the Caché Classes.

This template can be used by CREATE^CASH to create multiple Classes for the same FileMan File, using different Flags for different purposes.

If the "C" Flag is passed to START^CASH, then CREATE^CASH will be called automatically.

Format

Set RETURN=\$\$START^CASH(FILE,FLAGS,PACKAGE,ID,OWNER,.LIST)

Or

Do START^CASH(FILE,FLAGS,PACKAGE,ID,OWNER,.LIST)

Note: The OWNER parameter is new! It has been inserted before the LIST parameter, which is passed by reference, to try and prevent LIST being passed as null. The \$D() check sees this as a LIST array being passed and maps no fields other than the ID, IENS and .01 field!

Parameters

FILE	(Required) The Fileman File # to be Discovered	
FLAGS	(Optional) Flags listed below:	
	C	Compile - Create & Compile the Cache classes after the file(s) have been Discovered. [NOTE: The CREATE^CASH call will be executed with all other parameters passed straight through if "C" is passed in here. If "C" is not passed, all parameters other than FILE and the "V" Flag are ignored by START^CASH!
	D	Descriptions – Adds the full field description from ^DD to the Property Description. This data does not need to be stored in file #15050.11, as it is just copied directly at compile time, and is therefore a compile only option (FLAGS["C"]) . Adding the full description means that this text is viewable in the HTML class documentation!

	E	<p>Expand - Pointers will generate a Computed Property that expands the .01 field (by default) in the pointed to file.</p> <p>[NOTES:</p> <ol style="list-style-type: none"> 1. Can be used in conjunction with the "P" flag. 2. If you want to expand a field other than the .01 default, generate the Class, amend the PFIELD parameter of the Computed Property and then re-compile.
	F	<p>Force - Force the creation and compilation of a Class even if it already exists. The old version of the Class will be deleted!</p>
	I	<p>Simple IDs – Use a simple ID rather than one generated from the Class name. "IEN" will be used, unless an alternative ID is passed (e.g. ID="RowID").</p>
	L	<p>Loose Validation – This flag relaxes the constraints placed on Properties, so exporting data to a 3rd party SQL database should be easier. You are less likely to get import errors when the FileMan data does not meet it's own constraints (e.g. nulls in a required field, or an invalid date in a Date/Time field).</p>
	M	<p>Multiples - Generate Sub-Classes for Multiple fields and create the appropriate Parent-Child Relationships.</p>
	N	<p>Simple Names - Use simpler names, i.e. the file name concatenated with file #. Multiples won't get parent and grandparent names prefixed.</p>
	P	<p>Pointers - Generate Classes for pointed to files and create the appropriate Foreign Keys.</p> <p>[NOTE: Can be used with the "E" flag]</p>
	Q	<p>SQL Only Compile – Only tables are compiled, not full classes. Caché Classes generally provide multiple types of access (e.g. SQL and Object). Using the "Q" flag restricts acces to SQL only, so objects cannot be instantiated and classmethods cannot be called. This flag is useful if you know you only need SQL access, or want to restrict access to SQL-only.</p>
	R	<p>Recursive - Multiples and Pointers will also generate Sub-Classes and Classes for their Multiples and Pointers.</p> <p>[NOTE: If neither "R" nor "r" are passed, "P" will be stripped from FLAGS when calling CREATE ^CASH to create Sub-Classes, and both "M" and "P" will be removed when creating Pointer Classes.</p> <p>If "R" is passed, "M" and "P" will remain in FLAGS for all Sub-Classes and Pointer Classes.]</p> <p>[WARNING: Using "R" can generate lots of Classes!]</p>

	r	<p>Partially Recursive – Multiples will generate Classes for any Pointer Fields, but those Pointers will not create Sub-Classes or Classes for Multiples or Pointers that they contain.</p> <p>[NOTE: If "r" is passed "P" will remain in FLAGS for Sub-Classes, but both "M" and "P" will be stripped from Pointer Classes. If "R" is passed it will override "r"!]</p>
	S	<p>SOAP Web Services - Create a new sister Class inherited from %SOAP.WebService. This Class contains SQL Stored Procedures generated for each index in the main Class (plus the default ByID Procedure). These Queries are also exposed as Web Methods. You can invoke a web test page with the following url:</p> <p>http://{servername}/csp/{namespace}/{package}.{class}WS.cls</p> <p>You can view the WSDL Service Description by appending "?WSDL=1" to this url.</p> <p>For example:</p> <p>http://localhost/csp/vista/User.PatientWS.cls http://localhost/csp/vista/User.PatientWS.cls?WSDL=1</p> <p>[NOTE: Version 5.0.* and later only!]</p>
	V	<p>Verbose - The default is Silent Mode (though the Caché compile messages will print to screen either way).</p>
	W	<p>Web Page - Add the %CSP.Page Super Class to the generated Classes ("X" must also be specified). The OnPage() method will be overridden to display the output of the XMLDump() method.</p> <p>[NOTE: Version 5.0.* and later only!]</p>
	X	<p>XML - Add the %XML.Adaptor Super Class to each of the generated Classes. An XMLDump() method will be created to export the entire file in XML format. This uses the inherited XMLExport() instance method.</p> <p>[NOTE: Version 5.0.* and later only!]</p>
PACKAGE		<p>(Optional) The Caché Package for generated Classes. If not passed in, the default Package "User" will be used, which has an associated SQL schema of "SQLUser". The Package name cannot be a SQL Reserved Word.</p>
ID		<p>(Optional) This value overrides the default ID string of "IEN" for each Class, if Simple IDs are requested by passing in the "I" flag.</p>
OWNER		<p>(Optional) A valid Caché SQL Username. The classes will be created with this user as the owner. If null, the default is "_System".</p>

LIST	<p>(Optional) Array of fields to include in the mapped Class(es). All other Fields will be ignored if this array is passed.</p> <p>[NOTE: If LIST is not passed, all fields will be mapped!]</p> <p>The LIST should be in the format:</p> <p>LIST(file#,field#1)=""</p> <p>...</p> <p>LIST(file#,field#n)=""</p> <p>Only the specified fields will be added to the Class (plus any required keys).</p> <p>To specify fields in Multiples or Pointers, use the relevant Flags ("M" or "P") and add array nodes as follows:</p> <p>LIST(multiple_file#,field#1)=""</p> <p>...</p> <p>LIST(multiple_file#,field#n)=""</p> <p>LIST(pointer_file#,field#1)=""</p> <p>...</p> <p>LIST(pointer_file#,field#n)=""</p> <p>Example:</p> <p>LIST(4,.01)="" - .01 field (Name) for file# 4 (Institution)</p> <p>LIST(4,.02)="" - .02 field (State) pointer to file #5</p> <p>LIST(5,.01)="" - .01 field (Name) for file# 5 (State)</p> <p>LIST(5,3)="" - 3 field (County) multiple file# 5.01</p> <p>LIST(5.01,.01)="" - .01 field (County) for file #5.01</p> <p>This produces 3 Classes, assuming the "M" and "P" Flags are passed, each with one data Property, plus the relevant Relationships and Foreign Keys. If the "E" Flag is also passed a Computed Field will also be generated for the Pointer.</p>
-------------	---

Return Value

1	Success
0	Failure

Examples

Example 1: Discover a FileMan File.

```
Do START^CASH(5,"V")
5: State
5.01: StateCounty
5.02: StateCountyZipCode
```

Note: Any Multiples (and Pointers) are automatically Discovered as well!
The following has the same effect but without feedback:

```
Write $$START^CASH(5)
1
```

Note: No Cache Classes have been created or compiled!

Example 2: Discover a FileMan File and create a Caché Class in the "HDIS" package.

```
Write $$START^CASH(2,"CF","HDIS")
Compilation started on 05/44/2006 16:06:55
Compiling class HDIS.Patient .....
Compiling table HDIS.Patient ...
Compiling routine HDIS.Patient.1
Compiling routine HDIS.Patient.2
Compiling routine HDIS.Patient.3
Compiling routine HDIS.Patient.4
Compiling routine HDIS.Patient.5
Compiling routine HDIS.Patient.6
Compiling routine HDIS.Patient.7
Compilation finished successfully.
1
```

Note: The Patient File (#2) is large and creates more than one routine!

Example 3: Discover a FileMan File, its Sub-Files and Pointers and create the Caché Classes.

```
Write $$START^CASH(22,"CFMEP")
Compilation started on 05/24/2006 16:07:03
Compiling class User.PeriodOfService .....
Compiling table SQLUser.PeriodOfService ...
Compiling routine User.PeriodOfService.1
Compilation finished successfully.

Compilation started on 05/24/2006 16:07:03
Compiling class User.PowPeriod .....
Compiling class User.PowPeriodSynonym .....
Compiling table SQLUser.PowPeriod ...
Compiling table SQLUser.PowPeriodSynonym ...
Compiling routine User.PowPeriod.1
Compiling routine User.PowPeriodSynonym.1
Compilation finished successfully.
1
```

Note: File #22 is PowPeriod. It points to file #21, PeriodOfService, which is created and compiled first. It also has a Multiple, file #22.01 called Synonym, which is created and compiled with it as PowPeriodSynonym.

Also, the default "User" package is used as I didn't specify otherwise, but notice the Tables are in the SQLUser Schema! This is because "User" is a SQL Reserved Word.

Example 4: Use the LIST array to create a couple of small Classes.

Note: This is a complex example and introduces some advanced techniques.

```
Set LIST(4,.01)=" "
Set LIST(4,.02)=" "
Set LIST(5,.01)=" "
Set LIST(5,3)=" "
Set LIST(5.01,.01)=" "
Do START^CASH(4,"CFMEPRV","HDIS","",".LIST)

4: Institution
5: State
5.01: StateCounty
5.02: StateCountyZipCode
4.03: InstitutionContact
44: HospitalLocation
40.9: LocationType
```

```
40.8: MedicalCenterDivision
40.801: MdclCntrDvsnEmplyHlthCnts
```

[...There are 867 Files Discovered here!]

```
4.1: FacilityType
4.014: InstitutionAssociations
4.05: InstttnAssctnTypes
4.01: InstitutionPackageXref
```

```
Deleting class HDIS.Institution           done.
Deleting class HDIS.State                 done.
Deleting class HDIS.StateCounty          done.
Compilation started on 05/24/2006 16:22:55
Compiling class HDIS.State .....
Compiling class HDIS.StateCounty .....
Compiling table HDIS.State ...
Compiling table HDIS.StateCounty ...
Compiling routine HDIS.State.1
Compiling routine HDIS.StateCounty.1
Compilation finished successfully.

Compilation started on 05/24/2006 16:22:57
Compiling class HDIS.Institution .....
Compiling table HDIS.Institution ...
Compiling routine HDIS.Institution.1
Compilation finished successfully.
```

Note: The Institution file (#4) has many Pointers and Multiples, but only State (#5) is in LIST, so only State and StateCounty are created and compiled. The County Sub-file of State is mapped because the "R" Flag was passed in to recursively map Pointers and Multiples.

WARNING: Because the "R" Flag was passed all Multiples and Pointers were recursively checked, resulting in 867 files being Discovered! If the LIST array hadn't been passed in, all 867 Classes would have been created and compiled!

An alternative way to achieve the same result would be as follows:

```
Set LIST(5,.01)=" "
Set LIST(5,3)=" "
Set LIST(5.01,.01)=" "
Do START^CASH(5,"CFMV","HDIS","",.LIST)

5: State
5.01: StateCounty
5.02: StateCountyZipCode

Deleting class HDIS.State           done.
Deleting class HDIS.StateCounty     done.
Compilation started on 05/24/2006 16:40:55
Compiling class HDIS.State .....
Compiling class HDIS.StateCounty .....
Compiling table HDIS.State ...
Compiling table HDIS.StateCounty ...
Compiling routine HDIS.State.1
Compiling routine HDIS.StateCounty.1
Compilation finished successfully.
```

Note: The "M" Flag means the ZipCode Sub-file (#5.02) of the County Sub-file (#5.01) will be discovered, though it isn't created or compiled because of the LIST array. The "F" flag is used here to force the deletion and re-creation of the State and StateCounty classes.

```

Set LIST(4,.01)=" "
Set LIST(4,.02)=" "
D $SYSTEM.OBJ.Delete("HDIS.Institution")

Deleting class HDIS.Institution                                done.
Do START^CASH(4,"CEPV","HDIS","",.LIST)

4: Institution
5: State
5.01: StateCounty
5.02: StateCountyZipCode
4.03: InstitutionContact
4.1: FacilityType
4.014: InstitutionAssociations
4.01: InstitutionPackageXref
4.2: Domain
4.21: DomainTransmissionScript
4.22: DomainTrnsmssnScriptText
4.299: DmnTrnsScrpTrnsScrpNts
4.25: DomainNotes
4.23: DomainSynonym
4.24: DomainPollList
4.11: Agency

Compilation started on 05/24/2006 16:43:26
Compiling class HDIS.Institution .....
Compiling table HDIS.Institution ...
Compiling routine HDIS.Institution.1
Compilation finished successfully.

```

Note: We need to manually delete HDIS.Institution because we are not going to pass the "F" flag this time! If we did pass "F", we would overwrite the State Class that we had just created, and it would no longer be linked to StateCounty, as the "M" flag wouldn't be used for State without passing "R".

There are still 16 files Discovered, as they are referenced by either Multiples or Pointers in the Institution file. Only the small Institution Class is created though, which contains the .01 field (Name) and the .02 field (State). We actually get two Properties for State as we passed the "E" and "P" flags, an expanded Computed field (State) and a Reference to the HDIS.State Class (StateID).

Note: Files are discovered very quickly, so the above isn't really a performance overhead. The discovery process gathers the same information whatever flags are passed in, so you can use CREATE^CASH to compile different versions of these Classes afterwards.

The "r" flag can also be useful for scenarios like Example 4.

3.2 CREATE^CASH

This call can be used to create or re-create any File that has been discovered by START^CASH. This can be useful if you want to create several versions of a Class in different Packages. You may, for example want to create the following types of Classes for a particular file:

- A full Class, with all Pointers and Multiples ("M" and "P" flags)
- A "flattened" Class, with Pointers expanded as Computed Fields ("E" flag)
- A lightweight Class with a handful of specific fields (LIST array)

Format

Set RETURN=\$\$CREATE ^CASH(FILE,FLAGS,PACKAGE,ID,OWNER,.LIST)

Or

Do CREATE^CASH(FILE,FLAGS,PACKAGE,ID,OWNER,.LIST)

Note: The OWNER parameter is new! It has been inserted before the LIST parameter, which is passed by reference, to try and prevent LIST being passed as null. The \$D() check sees this as a LIST array being passed and maps no fields other than the ID, IENS and .01 field!

Parameters

FILE	(Required) The Fileman File# of the Class to be Created and Compiled. This File # must have been discovered and have an entry in File# 15050.11.	
FLAGS	(Optional) Flags listed below:	
	D	Descriptions – Adds the full field description from ^DD to the Property Description. This data does not need to be stored in file #15050.11, as it is just copied directly at compile time, and is therefore a compile only option. Adding the full description means that this text is viewable in the HTML class documentation!
	E	Expand - Pointers will generate a Computed Property that expands the .01 field (by default) in the pointed to file. [NOTES: 1. Can be used in conjunction with the "P" flag. 2. If you want to expand a field other than the .01 default, generate the Class, amend the PFIELD parameter of the Computed Property and then re-compile.
	F	Force - Force the creation and compilation of a Class even if it already exists. The old version of the Class will be deleted!
	I	Simple IDs – Use a simple ID rather than one generated from the Class name. "IEN" will be used, unless an alternative ID is passed (e.g. "RowID").
	L	Loose Validation – This flag relaxes the constraints placed on Properties, so exporting data to a 3 rd party SQL database should be easier. You are less likely to get import errors when the FileMan data does not meet it's own constraints (e.g. nulls in a required field, or an invalid date in a Date/Time field).
	M	Multiples - Generate Sub-Classes for Multiple fields and create the appropriate Parent-Child Relationships.
	N	Simple Names - Use simpler names, i.e. the file name concatenated with file #. Multiples won't get parent and grandparent names prefixed.

	P	<p>Pointers - Generate Classes for pointed to files and create the appropriate Foreign Keys. [NOTE: Can be used with the "E" flag]</p>
	Q	<p>SQL Only Compile – Only tables are compiled, not full classes. Caché Classes generally provide multiple types of access (e.g. SQL and Object). Using the "Q" flag restricts acces to SQL only, so objects cannot be instantiated and classmethods cannot be called. This flag is useful if you know you only need SQL access, or want to restrict access to SQL-only.</p>
	R	<p>Recursive - Multiples and Pointers will also generate Sub-Classes and Classes for their Multiples and Pointers. [NOTE: If neither "R" nor "r" are passed, "P" will be stripped from FLAGS when calling CREATE ^CASH to create Sub-Classes, and both "M" and "P" will be removed when creating Pointer Classes. If "R" is passed, "M" and "P" will remain in FLAGS for all Sub-Classes and Pointer Classes.] [WARNING: Using "R" can generate lots of Classes!]</p>
	r	<p>Partially Recursive – Multiples will generate Classes for any Pointer Fields, but those Pointers will not create Sub-Classes or Classes for Multiples or Pointers that they contain. [NOTE: If "r" is passed "P" will remain in FLAGS for Sub-Classes, but both "M" and "P" will be stripped from Pointer Classes. If "R" is passed it will override "r"!]</p>
	S	<p>SOAP Web Services - Create a new sister Class inherited from %SOAP.WebService. This Class contains SQL Stored Procedures generated for each index in the main Class (plus the default ByID Procedure). These Queries are also exposed as Web Methods. You can invoke a web test page with the following url: <code>http://{servername}/csp/{namespace}/{package}.{class}WS.cls</code> You can view the WSDL Service Description by appending "?WSDL=1" to this url. For example: http://localhost/csp/vista/User.PatientWS.cls http://localhost/csp/vista/User.PatientWS.cls?WSDL=1 [NOTE: Version 5.0.* and later only!]</p>
	V	<p>Verbose - The default is Silent Mode (though the Caché compile messages will print to screen either way).</p>
	W	<p>Web Page - Add the %CSP.Page Super Class to the generated Classes ("X" must also be specified). The OnPage() method will be overridden to display the output of the XMLDump() method. [NOTE: Version 5.0.* and later only!]</p>

	X	<p>XML - Add the %XML.Adaptor Super Class to each of the generated Classes. An XMLDump() method will be created to export the entire file in XML format. This uses the inherited XMLExport() instance method.</p> <p>[NOTE: Version 5.0.* and later only!]</p>
PACKAGE		<p>(Optional) The Caché Package for generated Classes. If not passed in, the default Package "User" will be used, which has an associated SQL schema of "SQLUser". The Package name cannot be a SQL Reserved Word.</p>
ID		<p>(Optional) This value overrides the default ID string of "IEN" for each Class, if Simple IDs are requested by passing in the "I" flag.</p>
OWNER		<p>(Optional) A valid Caché SQL Username. The classes will be created with this user as the owner. If null, the default is "_System".</p>
LIST		<p>(Optional) Array of fields to include in the mapped Class(es). All other Fields will be ignored if this array is passed.</p> <p>[NOTE: If LIST is not passed, all fields will be mapped!]</p> <p>The LIST should be in the format:</p> <p>LIST(file#,field#1)=""</p> <p>...</p> <p>LIST(file#,field#n)=""</p> <p>Only the specified fields will be added to the Class (plus any required keys).</p> <p>To specify fields in Multiples or Pointers, use the relevant Flags ("M" or "P") and add array nodes as follows:</p> <p>LIST(multiple_file#,field#1)=""</p> <p>...</p> <p>LIST(multiple_file#,field#n)=""</p> <p>LIST(pointer_file#,field#1)=""</p> <p>...</p> <p>LIST(pointer_file#,field#n)=""</p> <p>Example:</p> <p>LIST(4,.01)="" - .01 field (Name) for file# 4 (Institution)</p> <p>LIST(4,.02)="" - .02 field (State) pointer to file #5</p> <p>LIST(5,.01)="" - .01 field (Name) for file# 5 (State)</p> <p>LIST(5,3)="" - 3 field (County) multiple file# 5.01</p> <p>LIST(5.01,.01)="" - .01 field (County) for file #5.01</p> <p>This produces 3 Classes, assuming the "M" and "P" Flags are passed, each with one data Property, plus the relevant Relationships and Foreign Keys. If the "E" Flag is also passed a Computed Field will also be generated for the Pointer.</p>

Return Value

1	Success
0	Failure

Examples

For the following examples lets assume that I discovered File#22 earlier, perhaps by running this command:

```
Do START^CASH(22,"MPV")

22: PowPeriod
22.01: PowPeriodSynonym
21: PeriodOfService
21.01: PeriodOfSrvceElgblty
21.02: PeriodOfServiceSynonym
```

Example 1: Create a set of Classes with full functionality

```
Write $$CREATE^CASH(22,"CFMEPVXWS","HDISFull")

Compilation started on 05/24/2006 15:11:14
Compiling class HDISFull.PeriodOfService .....
Compiling table HDISFull.PeriodOfService ...
Compiling routine HDISFull.PeriodOfService.1
Compiling routine HDISFull.PeriodOfService.2
Compilation finished successfully.

Compilation started on 05/24/2006 15:11:15
Compiling class HDISFull.PeriodOfServiceWS .....
Compiling routine HDISFull.PeriodOfServiceWS.1
Compiling routine HDISFull.PeriodOfServiceWS.2
Compiling class HDISFull.PeriodOfServiceWS.ByAbbreviation .....
Compiling class HDISFull.PeriodOfServiceWS.ByAbbreviation.DS .....
Compiling class HDISFull.PeriodOfServiceWS.ByCode .....
Compiling class HDISFull.PeriodOfServiceWS.ByCode.DS .....
Compiling class HDISFull.PeriodOfServiceWS.ByID .....
Compiling class HDISFull.PeriodOfServiceWS.ByID.DS .....
Compiling class HDISFull.PeriodOfServiceWS.ByName .....
Compiling class HDISFull.PeriodOfServiceWS.ByName.DS .....
Compiling routine HDISFull.PeriodOfServiceWS.ByAbbreviation.1
Compiling routine HDISFull.PeriodOfServiceWS.ByAbb.DS.1
Compiling routine HDISFull.PeriodOfServiceWS.ByCode.1
Compiling routine HDISFull.PeriodOfServiceWS.ByCod.DS.1
Compiling routine HDISFull.PeriodOfServiceWS.ByID.1
Compiling routine HDISFull.PeriodOfServiceWS.ByID.DS.1
Compiling routine HDISFull.PeriodOfServiceWS.ByName.1
Compiling routine HDISFull.PeriodOfServiceWS.ByNam.DS.1
Compilation finished successfully.

Compilation started on 05/24/2006 15:11:18
Compiling class HDISFull.PowPeriod .....
Compiling class HDISFull.PowPeriodSynonym .....
Compiling table HDISFull.PowPeriod ...
Compiling table HDISFull.PowPeriodSynonym ...
Compiling routine HDISFull.PowPeriod.1
Compiling routine HDISFull.PowPeriod.2
Compiling routine HDISFull.PowPeriodSynonym.1
Compilation finished successfully.

Compilation started on 05/24/2006 15:11:19
Compiling class HDISFull.PowPeriodWS .....
Compiling routine HDISFull.PowPeriodWS.1
Compiling routine HDISFull.PowPeriodWS.2
Compiling class HDISFull.PowPeriodWS.ByAbbreviation .....
Compiling class HDISFull.PowPeriodWS.ByAbbreviation.DS .....
```

```

Compiling class HDISFull.PowPeriodWS.ByID .....
Compiling class HDISFull.PowPeriodWS.ByID.DS .....
Compiling class HDISFull.PowPeriodWS.ByName .....
Compiling class HDISFull.PowPeriodWS.ByName.DS .....
Compiling routine HDISFull.PowPeriodWS.ByAbbreviation.1
Compiling routine HDISFull.PowPeriodWS.ByAbbreviat.DS.1
Compiling routine HDISFull.PowPeriodWS.ByID.1
Compiling routine HDISFull.PowPeriodWS.ByID.DS.1
Compiling routine HDISFull.PowPeriodWS.ByName.1
Compiling routine HDISFull.PowPeriodWS.ByName.DS.1
Compilation finished successfully.

Compilation started on 05/24/2006 15:11:21
Compiling class HDISFull.PowPeriodSynonymWS .....
Compiling routine HDISFull.PowPeriodSynonymWS.1
Compiling class HDISFull.PowPeriodSynonymWS.ByID .....
Compiling class HDISFull.PowPeriodSynonymWS.ByID.DS .....
Compiling routine HDISFull.PowPeriodSynonymWS.ByID.1
Compiling routine HDISFull.PowPeriodSynonymWS.ByID.DS.1
Compilation finished successfully.
1

```

As for Example 3 under START^CASH above, we get Classes created for PeriodOfService, PowPeriod and PowPeriodSynonym. If you look at them in Studio though you will see that they have two extra Super Classes: %XML.Adaptor and %CSP.Page. There are also two new methods XMLDump() and OnPage().

You will also see that a number of Classes ending with WS have been compiled. These are the SOAP Web Service Classes, and the Sub-Classes starting By... are created for each Web Method (Stored Procedure), the tool creates one for each unique index.

The following command will return 100 rows of data in XML format:

```

Do ##class(HDISFull.PowPeriod).XMLDump(100)

<PowPeriod>
  <Abbreviation>WWI</Abbreviation>
  <Name>WORLD WAR I</Name>
  <PowPeriodID>1</PowPeriodID>
  <SynonymID>
    <PowPeriodSynonym>
      <PowPeriodSynonymID>1</PowPeriodSynonymID>
      <Synonym>WWI</Synonym>
    </PowPeriodSynonym>
  </SynonymID>
</PowPeriod>
<PowPeriod>
...

```

The following link does the same, but in a browser (you need to insert your information):

<http://{servername}/csp/{namespace}/HDISFull.PowPeriod.cls?MAXROWS=100>

This next link will give you access to the SOAP WebServices:

<http://{servername}/csp/{namespace}/HDISFull.PowPeriodWS.cls>

[NOTE: XML, CSP and SOAP functionality is only available on Caché 5.0.* or later!]

Example 2: Create a simple “flattened” Class from the same discovery file

```
Write $$CREATE^CASH(22,"CE","HDISFlat")

Compilation started on 05/24/2006 15:12:03
Compiling class HDISFlat.PowPeriod .....
Compiling table HDISFlat.PowPeriod ...
Compiling routine HDISFlat.PowPeriod.1
Compilation finished successfully.

1
```

This doesn't create Sub-Classes for Multiples or Classes for Pointers. The Pointer Fields are expanded as Computed Properties in the main Class because the “E” flag was passed.

Example 3: Create a small Class with only 3 Properties from the same discovery file

```
Set LIST(22,.01)=" "
Set LIST(22,1)=" "
Set LIST(22,101)=" "

Write $$CREATE^CASH(22,"CE","HDISSmall","",.LIST)

Compilation started on 05/24/2006 15:13:50
Compiling class HDISSmall.PowPeriod .....
Compiling table HDISSmall.PowPeriod ...
Compiling routine HDISSmall.PowPeriod.1
Compilation finished successfully.

1
```

This maps only the .01,1 and 101 fields. The 101 field is a Pointer, so as the “E” flag was passed a Computed Property will be created for it.

3.3 ALL^CASH

Format

Do ALL^CASH(FLAGS,PACKAGE,ID,OWNER)

Parameters

FLAGS	(Optional) Flags “EFMPRV” are always passed to CREATE^CASH by ALL^CASH. You can add the following flags if you wish:	
	I	Simple IDs – Use a simple ID rather than one generated from the Class name. “IEN” will be used, unless an alternative ID is passed (e.g. “RowID”).
	N	Simple Names - Use simpler names, i.e. the file name concatenated with file #. Multiples won't get parent and grandparent names prefixed.
	Q	SQL Only Compile – Only tables are compiled, not full classes. Caché Classes generally provide multiple types of access (e.g. SQL and Object). Using the “Q” flag restricts acces to SQL only, so objects cannot be instantiated and classmethods cannot be called. This flag is useful if you know you only need SQL access, or want to restrict access to SQL-only.

	S	<p>SOAP Web Services - Create a new sister Class inherited from %SOAP.WebService. This Class contains SQL Stored Procedures generated for each index in the main Class (plus the default ByID Procedure). These Queries are also exposed as Web Methods. You can invoke a web test page with the following url:</p> <p>http://{servername}/csp/{namespace}/{package}.{class}WS.cls</p> <p>You can view the WSDL Service Description by appending "?WSDL=1" to this url.</p> <p>For example: http://localhost/csp/vista/User.PatientWS.cls http://localhost/csp/vista/User.PatientWS.cls?WSDL=1</p> <p>[NOTE: Version 5.0.* and later only!]</p> <p>[WARNING!] Using this flag for ALL^CASH will create a large number of extra classes (Caché creates a new class for each Webmethod). E.g. I got 45312 classes generated from 7810 Fileman files and sub-files!</p> <p>You may also receive <INSUFFICIENT CLASS MEMORY> errors because the lock table is full! To alleviate this problem increase the 'Generic Memory Heap' setting (under 'Memory' on the 'Advanced' tab in the Caché Configuration Manager). I doubled the default value of 2816 to 5632 for the above example.]</p>
	W	<p>Web Page - Add the %CSP.Page Super Class to the generated Classes ("X" must also be specified). The OnPage() method will be overridden to display the output of the XMLDump() method.</p> <p>[NOTE: Version 5.0.* and later only!]</p>
	X	<p>XML – Add the %XML.Adaptor Super Class to each of the generated Classes. An XMLDump() method will be created to export the entire file in XML format. This uses the inherited XMLExport() instance method.</p> <p>[NOTE: Version 5.0.* and later only!]</p>
PACKAGE		<p>(Optional) The Caché Package for generated Classes. If not passed in, the default Package "User" will be used, which has an associated SQL schema of "SQLUser". The Package name cannot be a SQL Reserved Word.</p>
ID		<p>(Optional) This value overrides the default ID string of "IEN" for each Class, if Simple IDs are requested by passing in the "I" flag.</p>

Examples

Example 1: Do ALL^CASH()

This creates and compiles every FileMan file, using the default "User" Package and "SQLUser" SQL Schema.

Example 2: Do ALL^CASH("", "HDIS")

This time all the Classes are created in the "HDIS" Package and SQL Schema.

Example 3: Do ALL ^CASH("NI","HDIS","RowId")

This call uses the alternative naming convention for Classes, and the Unique Id for each Class is overridden as "RowId". The Classes are created in the "HDIS" Package and SQL Schema.

WARNING: Doing any of these calls can take a very long time on Caché 4.1.*!

3.4 Editing Classes

The generated Classes say "DO NOT EDIT" at the top and there are good reasons for not doing so, not least of these is that subsequently re-generating the Class will overwrite any changes you made!

Advanced users will feel comfortable tweaking the generated Classes though, and there are a couple of areas where this may be beneficial or even necessary.

NOTE: In Version 1.02 I plan to add another file that can hold User Preferences for particular files. Long term this will be a better mechanism to use, as you will be able to re-generate Classes repeatedly without losing these changes/preferences.

Expanded Pointer Fields ("E" flag)

If I have a field that is a Pointer to a File, for example File# 5 (State), and I pass in the "E" flag, I will get a Property generated like this:

```
Property State As CASH.FileMan.Pointer(PFIELD = .01, PFILE = 5)
```

The CASH.FileMan.Pointer Custom Datatype contains generated LogicalToDisplay and LogicalToOdbc methods, so if I look at this field using SQL via ODBC or by using the %EXTERNAL function I will get the expanded version.

For example I might get "ALABAMA" returned instead of the State IEN of 1.

The default field used by the Mapping Tool to expand CASH.FileMan.Pointer Datatypes is the .01 field, which in this case is the Name. This will usually be right, but what if I'd rather have the two-letter State code returned instead? This data is held in Field# 1 (Abbreviation) of the State File.

To change the behavior of my State field, I just need to amend the Property definition as follows:

```
Property State As CASH.FileMan.Pointer(PFIELD = 1, PFILE = 5)
```

The Class can then be recompiled. The same SQL will now return "AL" instead of "ALABAMA" for a State IEN of 1.

Output Transforms

The CASH.FileMan.String and CASH.FileMan.StringDateTime Custom Datatypes use the TRANSFORM parameter. This determines whether any Output Transform defined for a Field is used or ignored. You can change the way the Class behaves on a Field by Field basis by changing this parameter.

For example, if I have a Free Text field, I will have a generated Property like:

```
Property SomeText As CASH.FileMan.String(COLLATION = "EXACT", ..., TRANSFORM = 1)
```

If this Field does have an Output Transform, then the LogicalToDisplay() and LogicalToOdbc() methods will attempt to expand the Logical value (held in the global), by making a wrapped call to EXTERNAL^DILFD.

Sometimes these Transforms are unhelpful though, and you may want to turn them off for particular fields. To do this, change the value of TRANSFORM to 0 and re-compile the Class:

```
Property SomeText As CASH.FileMan.String(COLLATION = "EXACT", ..., TRANSFORM = 0)
```

NOTE: The default for CASH.FileMan.String is TRANSFORM = 1, whilst the default for CASH.FileMan.StringDateTime is TRANSFORM = 0. The tool knows how to convert date/times to ODBC format and display format (using standard FileMan calls). There is no guarantee that an Output Transform will produce the correct format, as Transforms are used for many purposes, so it is safer not to use them. You can always override this default and re-compile the class.

CASH.FileMan.Date and CASH.FileMan.DateTime do NOT use the TRANSFORM parameter, though it is specified as 0. Output Transforms are never executed for these Datatypes. If you have a Property of one of these types and you want it to use its Transform, you will have to change it to a CASH.FileMan.StringDateTime and change TRANSFORM = 1.

CASH.FileMan.SetOfCodes does have a TRANSFORM parameter, but the default is 0. It should only be used for the very rare cases where the data stored in the global needs to be transformed before it matches the internal value for the Set Of Codes. Yes, I have found examples of this!

CASH.FileMan.Numeric, CASH.FileMan.Pointer and CASH.FileMan.VariablePointer do not use TRANSFORM, and the parameter is not specified. Transforms would only interfere with the correct operation of these Datatypes.

WARNING: Some Output Transforms require other instance data to work (about 7% on my system). It is not possible to do this with the generated Classmethods, as they only have access to the Properties instance data, passed in as a parameter. The interfaces for these Classmethods are fixed and used internally by Caché, so I can't add extra parameters.

For now I try to identify these fields and ensure that TRANSFORM is set to 0. It may be possible to create a computed field instead that uses the Transform in the Get() method and SQL Compute Code. I may add this as a future option if requested.

4. Caveats and Warnings

4.1 System Performance

As with any extraction/reporting tool you need to exercise caution and restraint. Caché SQL is fast, and this tool creates classes very rapidly, but you must still be very careful!

For example, allowing users to perform commands like "SELECT * FROM VERY_LARGE_TABLE" on a live system is always asking for trouble!

4.2 Security Considerations

Caché SQL allows you to set up appropriate Users and Roles for your Schemas. You can do this using standard SQL commands such as CREATE USER, ALTER USER, DROP USER, GRANT and REVOKE, or you can use the SQL Manager Utility.

I would highly recommend that you do so, as the default Username (_System) and Password (sys) are not secure.

This tool does not do any of this for you! Please refer to the InterSystems documentation.

4.3 Database Size and Journaling

If you are creating a large number of Classes you will need to ensure that you have enough space! I have provided the figures from my test systems below, which will hopefully give you an idea of the impact on your system:

5315 FileMan Files/ Caché Classes	Before		After	
Caché Version	Size (MB)	Percent Full	Size (MB)	Percent Full
4.1.16	900	85%	4,000	89%
5.0.20	900	85%	1,700	81%

There could also be an impact on journaling. Running ALL^CASH and creating 5315 Classes created about 3Gb of journal files. You may want to stop journaling ^oddDEF and ^oddCOM if this is a problem, and perhaps ^CASH.

4.4 Re-Installation of Custom Datatypes (4.1.*)

On Caché version 4.1.* re-installing the Custom Datatypes, either by upgrading or just re-running CASHI, can be very slow if you have a very large number of Classes already in the Namespace (i.e. thousands). This is because the Caché compiler does a complete scan of the ^oddDEF global (where the class definitions are stored), presumably to build a list of dependencies.

NOTE: Please see Appendix A for more background on the Caché compiler in version 4.1.* and possible workarounds.

4.5 Running ALL ^CASH on Caché 4.1.*

On Caché 5.0.* this process took about 1 hour 22 minutes to create and compile 5315 Classes. On Caché 4.1.* compiling the same Classes took 47 hours 8 minutes! Both call the same system call (\$SYSTEM.OBJ.Compile). If you were looking for reasons to upgrade, this is one!

NOTE: These tests were done on default installations. The version 4.1.* compile suffered from "memory washout", which can be alleviated by increasing the global buffers. Doing this I have managed to cut this compile time to about 2 hours 30 minutes! See Appendix A for more detail on this and possible workarounds.

4.6 Very Large FileMan Files (250+ Fields)

4.6.1 Maximum Fields Per Class (250)

There is a limitation on the size of a Class in Caché (both 4.1.* and 5.0.*). This is largely because the Class Descriptor is %String and therefore limited to 32KB.

This means that Caché can have problems compiling Classes with more than 250-300 Properties. The number of Properties you can actually get away with will depend on a number of factors, such as name lengths, the number of methods, etc.

I found that none of my Files with more than 250 Fields would compile though. This affected 9 out of my 7400 Files, and one of these had over 2600 Fields!

My solution for this is to limit the number of Properties I add to a Class to 250, and to then put all remaining Properties into a continuation Class (or Classes).

I prioritize the Fields selected based on various factors including the FLAGS passed in. This is my prioritization order, with comments/assumptions in parentheses:

1. A Parent Reference (This is absolutely required for any Sub-Class).
2. The .01 Field (This is usually the most important bit of data).
3. Any Child References (Only if the "M" flag is passed, these are then required).
4. Any Indexed Fields (I'm assuming the Fields have been indexed because they are important - It also means the Class should be fully indexed).
5. Any Pointer Fields (Only if the "P" or "E" flags are passed – if both are passed then each Field counts twice!)
6. All other Fields in numerical order (I'm assuming Fields with lower numbers are more important!)

You can see that the combinations of "M", "P" and "E" flags passed can result in very different collections of Properties being included in the Class.

If you don't like any of those combinations, you can also use the LIST array to make your own.

As an example, if file #200 (NEW PERSON) has 600 fields, you could expect User.NewPerson to hold the most important fields, with User.NewPersonA and User.NewPersonB containing the rest.

[NOTE: The discovery file holds all of the Fields, even if there are over 250, so you can still call CREATE ^CASH multiple times with different flags and get the same Classes as you would had you run START ^CASH]

4.6.2 Maximum Multiples per Class (75)

I have also found that files with a very large number of Multiples or Word Processing Fields can cause compilation problems.

In these cases I limit the number of Sub-Classes to 75, with all others being moved to a continuation Class (or Classes).

This limitation can be imposed independently of, or in conjunction with, the Maximum Fields Per Class.

4.6.3 Caché Version 5.1

The 32KB Class Descriptor restriction has now been lifted! However, there are still issues with very large classes. You can get <CLASS TOO BIG TO COMPILE> or <ROUTINE TOO BIG TO COMPILE> errors, or even other more obscure problems.

For this reason, I have kept the above restrictions in place for this version, even though I originally planned to lift them.

4.6.4 CASH Parameter File (#15050.14)

The default limitation values are now held in this file:

`^CASH(15050.14,1,0)=MAXIMUM FIELDS PER CLASS^250`

`^CASH(15050.14,2,0)=MAXIMUM MULTIPLES PER CLASS^75`

You can change these defaults using the supplied APIs as follows:

- 1) `D UPDPARAM^CASHFN14(1,"",xxx)` - sets MAXIMUM FIELDS PER CLASS to xxx
- 2) `D UPDPARAM^CASHFN14(2,"",yy)` - sets MAXIMUM MULTIPLES PER CLASS to yy

If you need to change the values for an individual file (e.g. if a class still errors with the 250 limit, or if you want to try with more fields), you can create overrides as follows:

- 1) `D ADDOVR^CASHFN14(1,63.04,240)` – this overrides MAXIMUM FIELDS PER CLASS to 240 for File #63.04 only
- 2) `D ADDOVR^CASHFN14(2,63.04,70)` – this overrides MAXIMUM MULTIPLES PER CLASS to 70 for File #63.04 only

You can then use the `D UPDOVR^CASHFN14(1,63.04,230)` to update this value to 230, if 240 wasn't small enough for example.

`D DELOVR^CASHFN14(1,63.04)` will remove this override.

`D INIT^CASHFN14` will reset file #15050.14 to its default settings

WARNING: This call will remove all previously defined overrides!

Appendix A

The Caché 4.1. class compiler*

A “feature” of the Caché 4.1.* class compiler is that when a class is compiled a complete scan of the ^oddDEF global is made (where class definitions are held). Presumably this is to search for dependencies that are required for, or could affect, the compilation process.

This does not seem to be an issue with the Caché 5.0.* compiler, presumably because suitable indexes are used. The structure of ^oddDEF is completely different between the two versions. The Caché 5.0.* subscript names are less verbose, which may also help with size and performance.

The size of ^oddDEF affects compilation speed

An obvious side effect of this “feature” in 4.1.* is that as more classes are compiled, and ^oddDEF increases in size, it takes the compiler longer to scan this global. With several hundred classes present, the delay can become very noticeable (for a default installation).

ALL^CASH

When running ALL^CASH the above effect manifests itself very clearly. In a namespace that initially contains no classes, the compilation starts off very quickly, but noticeably slows down over time. Eventually the compilation pace slows to a crawl, which is why 5000+ classes can take 2 days! Compare this to just over 1 hour in Caché 5.0.*!

“Memory washout”

Caché, as the name implies, makes extensive use of memory cache to increase performance. When you scan a very large global, however, you can completely negate the effect of the cache, and adversely affect the performance of your whole system!

What follows isn’t supposed to be a detailed description, or treatise on Caché Memory Management, but will hopefully be enough of an overview to give the layman a rough understanding of what’s going on.

When global data is requested by a Caché process, the entire disk block containing that data is read into a section of memory referred to as a global buffer, but only if necessary, i.e. if it’s not already there! The Caché DBMS uses very intelligent algorithms to decide which blocks of data remain in memory and for how long. The more often that data is requested from a block which is already in memory the greater the performance gain!

The problem of “memory washout” can easily arise when you repeatedly scan the same very large global though. As you go through the nodes, each block needs to be read into memory, though for each individual scan none of the blocks are re-used! By the time you start scanning through the global a second time; all blocks need to be loaded into memory again. After a number of complete scans, all of the blocks in the global are being seen by Caché as often used, and hence given a higher priority to stay in cache longer. Eventually, the only blocks left in memory are likely to be

from this global, but paradoxically none are being re-used! Hence the effectiveness of cache is completely negated and you have "Memory Washout".

Incidentally, this is a good argument for having transactional systems and data mart/warehouse systems on different boxes (if you needed another reason). Transactional systems that mostly update a small amount of active data can run very quickly with a small memory footprint, using cache extremely efficiently. A reporting server is far more likely to require all the memory you can throw at it though!

Workarounds

So what can you do to prevent Memory Washout?

Well, here are two tactics (there may be others):

1. Make sure you have enough global buffers to hold the entire global!

Using a default Caché install with 16MB of global buffers, I was definitely getting "Memory Washout". The size of my ^oddDEF global got to be as large as 150MB when the tool had mapped all of the classes.

I found that raising the global buffers to 160MB pretty much removed the problem! Even if I already had hundreds of classes in my namespace, the first scan put them all in memory, so subsequent scans were very fast. The compiler barely slowed down at all, and I can now compile 7415 classes in under 2 hours 40 minutes. This is comparable to 5.0.* compile times. Interestingly the process made over 1.5 billion global references in this time!

Note: This is all very well if you are the only user on the system. If not, you will want to have more memory available for all the other processes! You may also want to combine this approach with the one below.

2. Reduce the amount of global buffers your process can take.

On a system with many users, you are definitely not going to want a "Memory Washout" situation! Even if the process running multiple scans has all of its data in cache, and its performance is OK, all other processes will crawl along, as they struggle to get their data into memory.

To remedy this, you can call the following before running your memory intensive process:

```
D LOW^%PRIO
```

This makes your process low priority (except on OpenVMS), and means it can only have access to a maximum of 25% of the total amount of global buffers available, leaving 75% for other processes.