



FileMan to Caché SQL Data Extraction and Mapping Tool

VERSION 1.0

Technical Manual and Package Security Guide

June 15, 2006

Veterans Health Administration
Office of Information
Health Data & Informatics

Revision History

Table 1, below, summarizes this document's revision history.

Date	Revision	Description	Author(s)
6/15/2006	1.0	Documentation created to support initial release.	Standard Terminology Services (STS)

Table 1: Technical Manual and Package Security Guide Revision History

Table of Contents



Revision History	ii
Table of Contents	iii
Table of Tables	iv
Using this Guide	v
1. Introduction	vi
2. Implementation and Maintenance	9
3. Files	11
4. Global Translation and Journaling	26
5. Routines	27
6. Exported Options	28
7. Exported RPCs	28
8. Other Software Elements	28
9. Callable Routines, Entry Points, and APIs	29
10. External Relations	40
11. Internal Relations	40
12. Software Product Security	41
13. Glossary	43

Table of Tables

<i>Table 1: Technical Manual and Package Security Guide Revision History</i>	<i>ii</i>
<i>Table 2: File Purging Routines</i>	<i>9</i>
<i>Table 3: Initial Disk Utilization Growth when 7400 FileMan Files are Mapped to Caché Classes</i>	<i>10</i>
<i>Table 4: FileMan Files Distributed with the CASH Tool</i>	<i>11</i>
<i>Table 5: Software Installation Requirements</i>	<i>40</i>
<i>Table 6: Security Keys and File Security</i>	<i>42</i>

Using this Guide

The following conventions are used in this document to indicate special information to the reader.

Symbol	Description
	Used to inform the reader of general information including references to additional reading material.
	Used to caution the reader to take special notice of critical information.

- Descriptive text is presented in a proportional font (as represented by this font).
- "Snapshots" of computer online displays (i.e., roll-and-scroll screen captures/dialogs) and computer source code are shown in a *non*-proportional font and enclosed within a box.
 - User responses to online prompts will be in boldface type and a non-proportional font.
 - The word "**Enter**" in snapshots further prompts the user to press the **Enter** or **Return** key on their keyboard.
 - Author comments are displayed in italics or as "callout" boxes.

1. Introduction

This technical manual describes the FileMan to Caché SQL Mapping Tool (CASH) version 1.0. This document is intended to assist Information Resources Management (IRM) and Enterprise VistA Support (EVS) staff.

This document provides a general overview of the data mapping process. This document does not attempt to provide a general overview of functionality in either FileMan or Caché, and expects that the reader has a working knowledge of these programs.

This FileMan to Caché SQL Mapping Tool has been designed to allow developers to create custom Caché SQL mappings for their FileMan Files quickly and easily. The tool relies on existing functionality available in both FileMan and Caché, and automates the creation of Caché Classes that are mapped to target FileMan files. The FileMan data is left unaltered, but is made available via Caché Classes, which provide support for SQL queries via ODBC and JDBC, among other options.

The tool has been designed to be lightweight and flexible and to have few dependencies outside of Caché and FileMan. The tool has been developed as a standard FileMan patch, and is released as a normal KIDS build.

This tool is being released in support of Data Standardization efforts. Data Standardization is supported by VHA Directive 2005-044. The tool will enable extraction of existing VistA reference data so that it can be thoroughly analyzed.

1.1 Data Mapping

The data mapping process involves two steps: FileMan file discovery, and Caché Class creation.

1.1.1 File Discovery

The Discovery process examines the FileMan data dictionary, extracting data for the target file and all related Multiples and Word Processing fields. All of the information required to create the corresponding Caché Class, plus all of the possible sub-Classes for the Multiples and Word Processing fields, is generated and stored in FileMan file #15050.11 (CASH FM CLASS MAP).

1.1.2 Caché Class Creation

The tool provides extensive control over the level of detail included in each Caché Class, and over the number and type of related Classes that are created, by using various flags and parameters. These flags and parameters are explained in detail in the “Callable Routines, Entry Points, and APIs” section of this manual.

Although the flags and parameters used have a direct effect on what Classes are created, they do not generally affect the Discovery process.

1.1.3 Caché Class Objects

You can choose how many files to map, and the level of detail to be included in the created

Classes. For example, Class compilation can include all discovered fields from a file, or only a selected list of fields.

Once Classes have been created by the compiler, they behave as normal Caché Class objects. This means that Caché support for database queries and application development can be leveraged on FileMan data.

For example, Caché 5.0 added better support for XML and Simple Object Access Protocol (SOAP), making it possible to expose FileMan data as XML or via a web service. These are options available for Caché 5.0 users only:

- 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 %CSP.Page super-Class can be added to the Class, with a generated OnPage() method that calls the XMLDump() method, so that the exported XML file is available via a web page.
- A “sister” Class can be created that uses the %SOAP.WebService super-Class. This enables SOAP access to the FileMan data via HTTP. Specifically, the sister Class will contain generated SQL stored procedures for each index in the target Class. A stored procedure is a database query that is preserved on the database server. The sister Class will expose those stored procedures as WebMethods, creating a link from the SOAP web service to the data mapped in the target Class.

1.2 Reference Materials

Readers who wish to learn more should consult the following:

- VA Data Standardization Project Intranet Website:
[REDACTED]
- The VistA Documentation Library (VDL) has more detailed information about all aspects of VistA. Readers may be especially interested in documentation about the MFS and Kernel Toolkit patches, which were released in support of the Data Standardization process (patch numbers XU*8.0*299 and XT*7.3*93). The VDL is available online:
[REDACTED]
- Caché is a product of InterSystems Corporation. More information about their product is available on their Website, <http://www.intersystems.com/>.
- Data Standardization is supported by VHA Directive 2005-044. A copy of this directive is available from [REDACTED]

Documentation is made available online, on paper and in Adobe Acrobat Portable Document Format (PDF). A PDF must be read using the Adobe Acrobat Reader (i.e., ACROREAD.EXE), which is freely distributed by Adobe Systems, Incorporated at the following Web address:
<http://www.adobe.com/>



For more information on the use of the Adobe Acrobat Reader, please refer to the "Adobe Acrobat Quick Guide," also available at the Adobe Web address above.



DISCLAIMER: The appearance of external hyperlink references in this manual does not constitute endorsement by the Department of Veterans Health Administration (VHA) of this Website or the information, products or services contained therein. The VHA does not exercise any editorial control over the information you may find at these locations.

2. Implementation and Maintenance

The following items constitute the recommended system maintenance for the CASH package.

2.1 Post Installation

After installation, POST^CASHI will run automatically to do the following.

1. Initialize file #15050.19 CASH SQL RESERVED WORDS with a list of reserved words recommended by InterSystems, in the Caché documentation.
2. Remove any old error log data from file #15050.13 CASH ERRORS.
3. Create and compile the utility Classes: CASH.FMFile, CASH.FMField and CASH.Utility.
4. Initialize file #15050.12 CASH CUSTOM DATATYPES.
5. Create and compile the custom datatype Classes (CASH.FileMan.*) .

2.2 Site Parameters

There are no site-specific parameters.

2.3 Archiving and Purging

There are no archiving capabilities in the CASH 1.0 package.

The following CASH files can be purged:

File Number	File Name	Purging Routine
15050.11	CASH FM CLASS MAP	D DELALL^CASHFN11
15050.12	CASH CUSTOM DATATYPES	D DELALL^CASHFN12
15050.13	CASH ERRORS	D DELALL^CASHFN13
15050.14	CASH PARAMETER FILE	D DELALL^CASHFN14
15050.19	CASH SQL RESERVED WORDS	D DELALL^CASHFN19 Warning: Purging this file on systems using Caché V 4.1 may result in SQL Reserved Word errors when compiling Classes!

Table 2: File Purging Routines

Purging these files is not required on a regular basis, and may not be needed at all. The following table identifies the amount of disk space required to map 7400 FileMan files on a test system. Re-mapping these files should not require significant additional disk space unless a large number of errors are generated. Note that disk space utilization will vary based on the number of FileMan files that are mapped to Caché Classes.

Caché Version	Growth (Mb)
4.1.16	4000
5.0.12	800

Table 3: Initial Disk Utilization Growth when 7400 FileMan Files are Mapped to Caché Classes

Disk utilization is not distributed evenly among the mapping tool's files. The majority of the above disk utilization comes from the Cache Class definitions themselves (e.g. ^oddDEF and ^oddCOM).

Caché has system utilities for purging created Classes:

D \$SYSTEM.OBJ.Delete(*ClassName*) can be used to purge individual Classes.*

D \$SYSTEM.OBJ.DeletePackage(*PackageName*) can be used to purge an entire package (group of Classes). See Caché's documentation for more information.

IMPORTANT: * It is recommended that you purge individual classes using the following CASH API though, as this will also remove all of the related classes that rely on the deleted class to function properly:

D CLDEL^CASHCU(*ClassName*)

Notes about Purging Individual Files

Growth in file #15050.11 is the most significant, but only occurs when files are being discovered by routines in the CASH tool. If a file has already been discovered and mapped, re-running a discovery routine will cause the original map to be overwritten, and this limits file size growth. Purging this file will have no effect on existing Caché Classes.

File #15050.12 should not grow at all. Purging it is harmless, but recovers little disk space.

Growth in file #15050.13 is expected to be minimal. Purging it should be done on an as-needed basis, if at all.

File #15050.19 contains data needed in order for this tool to be used with Caché 4.1 installations, and should only be purged if Caché 5.0 or later has been installed. If this file is purged on a system with Caché 4.1 installations, users may see SQL Reserved Word errors when compiling Classes.

3. Files

This section describes the data elements and VistA field locations needed included with the CASH tool.

3.1 Data Elements

The following files belong to the CASH tool. The following sections describe each file in more detail.

Number	Name	Description
15050.11	CASH FM CLASS MAP	<p>This file contains the maps (or templates) used to create Caché classes for FileMan files. Each map is created by the START^CASH call, which examines the ^DIC and ^DD structures for the file.</p> <p>This map can be used to create many different Caché Classes, with different options (FLAGS), by using the CREATE^CASH call.</p> <p>No data is distributed with this file.</p>
15050.12	CASH CUSTOM DATATYPES	<p>This file contains the maps (or templates) used to create the custom datatype Classes that support the CASH tool.</p> <p>No data is distributed with this file.</p>
15050.13	CASH ERRORS	<p>This file contains any error messages generated by the compile process, CREATE^CASH.</p> <p>No data is distributed with this file.</p>
15050.14	CASH PARAMETER FILE	<p>This file contains Parameters that can be used during the compile process, CREATE^CASH.</p> <p>No data is distributed with this file.</p>
15050.19	CASH SQL RESERVED WORDS	<p>This file contains a list of SQL Reserved Words. When the discovery process (START^CASH) assigns names, each one is checked against this list. If a match is found, an alternate name is stored for use in a SQL context. The alternate name is the original name with an underscore prefix (e.g. TABLE would be stored as _TABLE).</p> <p>No data is distributed with this file.</p>

Table 4: FileMan Files Distributed with the CASH Tool

3.1.1 CASH FM CLASS MAP File

File Name/Number: CASH FM Class Map file (#15050.11)

Global: ^CASH

System Location: Client (Facility)

This file contains the maps (or templates) used to create Caché Classes for FileMan files. Each map is created by the START^CASH call, which examines the ^DIC and ^DD structures for the file. This map can be used to create many different Caché Classes, by using the CREATE^CASH call with different options (FLAGS).

The following table contains the data elements being added to FileMan for this file:

Element		VISTA Field Location			
Description	File/Field	Name	Location	Data Type	Definition
Name Standardized Name for the Class to be created. If this Class is based on a top-level file, the name is taken from ^DIC. For Sub-files or WP Fields the name is a Standardized compound of this file's ^DD name, prefixed by the Parent file names. Note: The Standardizing process makes all words title case, then removes all spaces and other punctuation. The names are then shortened to 25 characters or less (required for uniqueness).	15050.11,.01	NAME	0;1	Free Text	3-30 Required
Sub Name Standardized name of this element only (not prefixed by parent file names), if this Class is based on a sub-file or WP Field. This field is left blank for top-level files. SUB NAME is used to name the Class if the "N" flag is used	15050.11,.02	SUB NAME	0;2	Free Text	3-30
SQL Name The name to be used for the SQL table (if different from NAME). SQL NAME must be specified if NAME is a reserved word in SQL.	15050.11,.03	SQL NAME	0;3	Free Text	3-30
File Type Type of the file this Class is based on (top-level, sub-file or WP Field). Used by the compiler to generate appropriate update code.	15050.11,.04	FILE TYPE	0;4	Set	F for File S for Sub-File W for WP Sub-File Required
Parent Field If this Class is based on a sub-file or WP Field, this field points to the related field in the parent file.	15050.11,.05	PARENT FIELD	0;5	Number	Between .0000001 and 99999999.
Properties Multiple containing Property details for the class.	15050.11,1	PROPERTIES	1;0	Multiple	1-30
Name Standardized name for the Property to be created.	15050.111,.01	NAME	0;1	Free Text	1-30 Required

Element		VISTA Field Location				
Description		File/Field	Name	Location	Data Type	Definition
Calculated Flag identifying whether or not the Property is calculated.		15050.111,.02	CALCULATED	0;2	Set	0 for No 1 for Yes Required
Cardinality Cardinality of a relationship type Property (one-to-many or parent-child).		15050.111,.03	CARDINALITY	0;3	Set	O for One M for Many P for Parent C for Children
Collection Type of collection Property.		15050.111;.04	COLLECTION	0;4	Set	A for Array L for List B for Binary Stream C for Character Stream
Inverse Inverse Class Name for a Relationship type Property.		15050.111,.05	INVERSE	0;5	Free Text	1-30
Relationship Is the Property a relationship?		15050.111,.06	RELATIONSHIP	0;6	Set	0 for No 1 for Yes
Required Is the Property required?		15050.111,.07	REQUIRED	0;7	Set	0 for No 1 for Yes Required
SQL Computed Is the Property a SQL computed field?		15050.111,.08	SQL COMPUTED	0;8	Set	0 for No 1 for Yes
Type The data type of the Property.		15050.111,.09	TYPE	0;9	Free Text	1-30 Required
Description The description of the Property, used in the Class definition.		15050.111,.11	DESCRIPTION	1;1	Free Text	1-80
SQL Compute Code SQL compute code for a SQL computed property.		15050.111,.21	SQL COMPUTE CODE	2;E1,245	Free Text	
Field Category Field category of the Property, used internally by the compiler.		15050.111,.31	FIELD CATEGORY	3;1	Set	C for child D for data M for multiple P for pointer T for transient W for multiple (wp)
SQL Field Name The SQL Field Name for the property (if different from Name).		15050.111,.32	SQL FIELD NAME	3;2	Free Text	3-30
Parameters Multiple containing Property parameters for the Class.		15050.111,1	PARAMETERS	1;0	Multiple	
	Parameter Name The name of the parameter.	15050.1111,.01	PARAMETER NAME	0;1	Free Text	1-30 Required
	Parameter Value The parameter's default value.	15050.1111,.02	PARAMETER VALUE	0;2	Free Text	1-30
	String Calculated field to display the parameter details.	15050.1111,1	STRING	1;0	M Code	

Element		VISTA Field Location				
Description		File/Field	Name	Location	Data Type	Definition
Methods Multiple containing Method details for the Class.		15050.11,2	METHODS	2;0	Multiple	
	Name The Method name.	15050.112,,01	NAME	0;1	Free Text	1-30 Required
	Class Method The Class method.	15050.112,,02	CLASS METHOD	0;2	Set	0 for No 1 for Yes Required
	Code Mode How the Method code is implemented (call, code, expression, generator, or object generator). Most Methods used by this tool will be a simple expression or a list of lines (code). The more complex datatype methods often use Method generators.	15050.112,,03	CODE MODE	0;3	Set	A for Call C for Code E for Expression G for Generator O for Objectgenerator
	Private Is the Method private?	15050.112,,04	PRIVATE	0;4	Set	0 for No 1 for Yes
	Return Type The datatype of the return value (if applicable).	15050.112,,05	RETURN TYPE	0;5	Free Text	1-30
	Field Number The FileMan field number associated with this Method (if applicable). If the Field being mapped is excluded from compilation, the Method will not be created.	15050.112,,06	FIELD NUMBER	0;6	Number	Between .0000001 and 99999999
	Description The description of the Method, used in the Class definition.	15050.112,,11	DESCRIPTION	1;1	Free Text	1-80
	Formal Spec The formal specification for this Method (a list of Parameters and their Datatypes).	15050.112,,21	FORMAL SPEC	2;1	Free Text	1-80
	Public List A list of public variables to be declared for this Method. This is only relevant for ProcedureBlock(=True) Methods, where variables need to be passed to called non-ProcedureBlock code.	15050.112,,31	PUBLIC LIST	3;1	Free Text	1-80
	Code Multiple containing the implementation code for the Method.	15050.112,1	CODE			
Maps Multiple containing storage map details for the Class.		15050.11,3	MAPS	3;0	Multiple	

Element		VISTA Field Location				
Description		File/Field	Name	Location	Data Type	Definition
Name Standardized name for the storage map to be created. Only one storage map per Class is defined by the CASH tool.		15050.113,.01	NAME	0;1	Free Text	1-30
Type The map type (data, index, bitmap, or bitmapextent). This tool will generate one Data Map per Class in the default "FileMan" Storage Map definition. All other Maps that are defined for this Class (if any) will be Indexes (of Type Index, Bitmap or Bitmapextent).		15050.113,.02	TYPE	0;2	Set	D for Data I for Index B for Bitmap E for Bitmapextent
Global Name The global that will be referenced by this storage map.		15050.113,.03	GLOBAL NAME	0;3	Free Text	1-8 Required
Subscripts Multiple containing the subscripts for this storage map.		15050.113,1	SUBSCRIPTS	1;0	Multiple	
Expression The expression to be used for this subscript in the storage map. This expression can be a string (e.g. "5" or "B"), a reference to a field with curly braces (e.g. "{Expression}"), or a reference to a FileMan file with a preceding "#" (e.g. "#5"). The expression refers to the STATE file Class. The name is not stored but rather is resolved at runtime, depending on the FLAGS and PACKAGE variables passed.		15050.1131,.01	EXPRESSION	0;1	Free Text	1-30 Required

Element			VISTA Field Location				
Description			File/Field	Name	Location	Data Type	Definition
		Loop Init Value The loop initialization value used by the storage map for this subscript, if applicable. FileMan data maps include a zero node that contains no relevant data for an individual row in Caché. Setting the LoopInitValue=0 instead of the default (null) causes this zero node to be skipped during creation of the Caché Class.	15050.1131,.02	LOOP INIT VALUE	0;2	Free Text	1-10
		Stop Expression The stop expression used by the storage map for this subscript, if applicable. FileMan data maps often contain index subscripts held at the same logical level as data. The index subscripts are non-numeric, e.g. "B," whereas the data-level subscripts are always numeric. Using a stop expression of "{Lx}" (where {Lx} is the appropriate subscript level variable) ensures that the loop quits when there are no more data rows.	15050.1131,.03	STOP EXPRESSION	0;2	Free Text	1-10
		Field Number The FileMan field number that this expression is associated with, where applicable. This field is used to identify a subset of FileMan Fields to map. When a Class is being compiled, only the specified subset of the fields is included in the Class.	15050.1131,.04	FIELD NUMBER	0;4	Number	Between .0000001 and 99999999.

Element			VISTA Field Location				
Description			File/Field	Name	Location	Data Type	Definition
		Alternate Expression An alternate expression for this expression, where applicable. This field is used for indexed Pointer fields. The alternate expression will usually have "ID" appended, e.g. "{State}" and "{StateID}". If FLAGS contains "E," then a map will be created for this Class, using the default expression. If FLAGS contains "P" a map will be created using the alternate expression. If FLAGS contains both "E" and "P", then both maps will be created; <i>but</i> , if FLAGS contains neither "E" nor "P" this storage map will <i>not</i> be created at all.	15050.1131,.05	ALTERNATE EXPRESSION	0;5	Free Text	3-30
		Data Multiple containing data elements for this storage map, where applicable. Index maps will not usually contain any data elements.	15050.113,2	DATA	2;0	Multiple	
		Name Standardized name for the data element. This must match a property name defined in file #15050.111.	15050.1132,.01	NAME	0;1	Free Text	1-30 Required
		Node The global node that contains this data element. A FileMan global reference is defined by the global name, the subscripts and this node, e.g. the STATE file (#5) is stored in ^DIC (Global Name), with two subscripts: 5 and the IEN, and the data is all contained in the '0' (zero) node. The full global reference would be: ^DIC(5,IEN,0).	15050.1132,.02	NODE	0;2	Free Text	1-30

Element			VISTA Field Location				
Description			File/Field	Name	Location	Data Type	Definition
		Piece The piece of the global node that contains this data element. It is assumed that the delimiter will be "^" and the \$Piece function will be used to extract the appropriate sub-string.	15050.1132,,03	PIECE	0;3	Number	Between 1-999, no decimal digits.
		Pointer Flag Is the data element a Pointer?	15050.1132,,04	POINTER FLAG	0;4	Number	0 or 1 Required
		Retrieval Code The retrieval code for this data element, where applicable. This field is used when NODE and PIECE are not defined and custom code must be used to extract the data.	15050.1132,,11	RETRIEVAL CODE	1;E1,245	Free Text	
Sub Files Multiple containing sub-files for this Class.			15050.11,4	SUB FILES	4;0	Pointer Multiple	Multiple containing Sub Files for this Class.
		Sub File Pointer to the sub-file.	15050.114,,01	SUB FILE	0;1	Pointer	Pointer to Cash FM Class Map File (#15050.11).

3.1.2 CASH CUSTOM DATATYPES File

File Name/Number: CASH Custom Datatypes (#15050.12)

Global: ^CASH

System Location: Client (Facility)

This file contains the maps (or templates) used to create the custom datatype classes that support the FM to Caché SQL Mapping tool.

The following table contains the data elements being added to FileMan for this file:

Element		VISTA Field Location			
Description	File/Field	Name	Location	Data Type	Definition
Name The name of the custom datatype to be created.	15050.12,.01	NAME	0;1	Free Text	3-30, not numeric and not starting with punctuation. Required
Client Data Type The Caché client datatype for this custom datatype. This is mandatory for Caché datatype classes, and specifies the Java/C++ or other client Class to which a variable is mapped.	15050.12,.02	CLIENT DATA TYPE	0;2	Free Text	1-20 Required
ODBC Type The Caché ODBC type for this custom datatype. This is the type used when the datatype is accessed via ODBC. The ODBC value must match the type in the OdbcToLogical() and LogicalToOdbc() Methods.	15050.12,.03	ODBC TYPE	0;3	Free Text	1-20 Required
SQL Category The Caché SQL category for this custom datatype. The data stored here maps the custom datatype to the appropriate Caché SQL datatype. Caché SQL relies on its datatypes to determine how to perform operations on a given type of data.	15050.12,.04	SQL CATEGORY	0;4	Free Text	1-20 Required
Include Generator An optional list of Caché include files (.INC), used when compiling any Method generator methods in this custom datatype.	15050.12,.05	INCLUDE GENERATOR	0;5	Free Text	1-30
Super An optional super Class for this custom datatype.	15050.12,.06	SUPER	0;6	Free Text	1-30
Description Word Processing Field containing description details for custom datatype. Used to document the Class.	15050.12,1	DESCRIPTION	1;0		
Methods Multiple containing details about the custom datatype methods.	15050.12,2	METHOD	2;0	Multiple	
Name The name of the Method.	15050.122,.01	NAME	0;1	Free Text	1-30 Required

Element		VISTA Field Location				
Description		File/Field	Name	Location	Data Type	Definition
Class Method Is the Method a Class Method? Datatype Methods will usually be Class Methods, rather than instance Methods.		15050.122,.02	CLASS METHOD	0;2	Number	0 for No 1 for Yes Required
Code Mode The code mode that the method uses when compiling. Code - this code is inserted directly into the Property Methods. Expression - the code is an extrinsic function called directly. Generator, objectgenerator - the code is generated at compile time, so each Property can have a unique version of this Method code (polymorphism).		15050.122,.03	CODE MODE	0;3	Set	C for Code E for Expression G for Generator O for Objectgenerator Required
Formal Spec The formal specification for the Method. This is a list of all the parameters, their type, default value and whether passed by value or reference. The specification's format is "Variable:datatype=default". Items in the list are separated by commas.		15050.122,.04	FORMAL SPEC	0;4	Free Text	1-80
Private Is the Method private?		15050.122,.05	PRIVATE	0;5	Number	0 for No 1 for Yes
Public List The public list for the Method. This contains a list of any variables that need to be declared as public in the Method. If the procedure block calls non-procedure block M code, you may need to declare some variables as public, so they can be used within this code.		15050.122,.06	PUBLIC LIST	0;6	Free Text	1-80
Return Type The datatype of the return value for this Method.		15050.122,.07	RETURN TYPE	0;7	Free Text	1-30
Description Word Processing Field containing description details for Method.		15050.122,1	DESCRIPTION	1;0	WP Field	
Code Word Processing Field containing code details for the Method.		15050.122,2	CODE	2;0	WP Field	
Parameters Multiple containing details about the custom datatype parameters.		15050.12,3	PARAMETERS	3;0	Multiple	

Element		VISTA Field Location				
Description		File/Field	Name	Location	Data Type	Definition
	Parameter Name The parameter name.	15050.123,.01	PARAMETER NAME	0;1	Free Text	1-30 Required
	Parameter Value The default value for the parameter. This can be overridden when defining the Property.	15050.123,.02	PARAMETER VALUE	0;2	Free Text	1-30
	Type The datatype of the Parameter, e.g. BOOLEAN.	15050.123,.03	TYPE	0;3	Free Text	1-50
	Description Word Processing field containing description details for the parameter.	15050.123,1	DESCRIPTION	1;0	?	?

3.1.3 CASH ERRORS File

File Name/Number: CASH Errors file (#15050.13)

Global: ^CASH

System Location: Client (Facility)

This file contains any error messages generated by the compile process, CREATE^CASH.

The following table contains the data elements being added to FileMan for this file:

Element		VISTA Field Location			
Description	File/Field	Name	Location	Data Type	Definition
Error Date/Time The date and time the error occurred.	15050.13,.01	ERROR DATE/TIME	0;1	Date	Required
Error Text The text of the error message returned.	15050.13,.02	ERROR TEXT	0;2	Free Text	1-80 Required
File Number The file number the CASH tool was processing at the time of the error.	15050.13,.03	FILE #	0;3	Number	Between .0000001 and 99999999.
Flags The FLAGS value(s) passed in by the mapping routine. Useful for debugging purposes.	15050.13,.04	FLAGS	0;4	Free Text	1-15
Package The PACKAGE value(s) passed in by the mapping routine. Useful for debugging purposes.	15050.13,.05	PACKAGE	0;5	Free Text	1-20
ID The ID value passed in by the mapping routine. Useful for debugging purposes.	15050.13,.06	ID	0;6	Free Text	1-20
Owner The OWNER value passed in by the mapping routine. Useful for debugging purposes.	15050.13,.07	OWNER	0;7	Free Text	1-20
List Multiple containing the values of LIST passed in for this compilation.	15050.13,1	LIST	1;0	Multiple	
List The value of the LIST array node passed in.	15050.131,.01	LIST	1;0	Free Text	1-30
\$ZE Error Text The text of the \$ZE system variable after the error. This may contain M code/syntax.	15050.13,2	\$ZE ERROR TEXT	2;0	Free Text	1-245 Required
Object Error Text The text of the special %objlasterror system variable. This will be in the format of %Library.Status, with embedded lists. Use \$\$SYSTEM.Status.DecomposeStatus() to interpret.	15050.13,3	OBJECT ERROR TEXT	3;E1,245	Free Text	1-245

3.1.4 CASH PARAMETER File

File Name/Number: CASH PARAMETER FILE (#15050.14)

Global: ^CASH

System Location: Client (Facility)

This file contains various parameters used by the CASH FM To Cache SQL Mapping Tool. The top level file contains the defaults used by the system, but these can be overridden in the OVERRIDES multiple.

The following table contains the data elements being added to FileMan for this file:

Element		VISTA Field Location				
Description		File/Field	Name	Location	Data Type	Definition
Parameter Name The Name used to identify the Parameter.		15050.14,.01	PARAMETER NAME	0;1	Free Text	3-30 Required
Parameter Value The default value for the named Parameter.		15050.14,.02	PARAMETER NAME	0;2	Free Text	1-30
Parameter Value The default value for the named Parameter.		15050.14,1	OVERRIDES	1;0	Multiple	
File Number This is the FileMan file (or sub-file) number that the Parameter value is to be overridden for. S:\$D(X) DINUM=X" is used in the Input Transform to ensure the value is keyed by the file #. NOTE: This field is NOT a Pointer to File #1, as we need to allow sub-files to be entered as well.		15050.141,.01	FILE NUMBER	0;1	Number	Between .000000001 and 9999999999 Required
Override Value The overridden value for the named Parameter for the file # specified		15050.141,.02	OVERRIDE VALUE	0;2	Free Text	0 for No 1 for Yes Required

3.1.5 CASH SQL RESERVED WORDS File

File Name/Number: CASH SQL Reserved Words file (#15050.19)

Global: ^CASH

System Location: Client (Facility)

This file contains a list of SQL reserved words. When the mapping process (START^CASH) assigns names, each one is checked against this list. If a match is found, an alternate name is stored for use in an SQL context. The alternate name is the original name with an underscore prefix (e.g. TABLE would be stored as _TABLE).



This file is not used in installations that have Caché version 5.0 or better. When the FileMan function \$\$VERSION^%ZOSV() shows that the Caché version is 5.0 or better, the system-provided call \$\$SYSTEM.SQL.IsReservedWord() is used instead. In these cases, file #15050.19 is installed, but is not used.

The following table contains the data elements being added to FileMan for this file:

Element	VISTA Field Location				
Description	File/Field	Name	Location	Data Type	Definition
Name The text of the SQL reserved word.	15050.19,.01	NAME	0;1	Free Text	2-30 Required

3.2 Package Default Definition

The following screen capture shows the CASH package's default definition.

PACKAGE: FM TO CACHE SQL 1.0

May 24, 2006 12:20 pm

PAGE 1

TYPE: SINGLE PACKAGE

TRACK NATIONALLY: YES

NATIONAL PACKAGE: FM TO CACHE SQL

DESCRIPTION:

FileMan To Cache SQL Mapping Tool - Version 1.0

ENVIRONMENT CHECK :

PRE-INIT ROUTINE : CLEAN^CASHI

POST-INIT ROUTINE : POST^CASHI

PRE-TRANSPORT RTN :

FILE #	NAME	UP DATE DD	SEND SEC. CODE	DATA COMES W/FILE	SITE DATA	RSLV PTS	USER OVER RIDE
15050.11	CASH FM CLASS MAP	YES	YES	NO			
15050.12	CASH CUSTOM DATATYPES	YES	YES	NO			
15050.13	CASH ERRORS	YES	YES	NO			
15050.14	CASH PARAMETER FILE	YES	YES	YES			
15050.19	CASH SQL RESERVED WORDS	YES	YES	NO			

4. Global Translation and Journaling

No file requires journaling.

There are no translation requirements.

5. Routines

The following routines are included in CASH:

CASH	CASHCN	CASHFN12	CASHUT02
CASH0	CASHCU	CASHFN13	CASHV4C
CASH1	CASHD	CASHFN14	CASHV4C0
CASH2	CASHDT01	CASHFN18	CASHV4C1
CASH3	CASHDT02	CASHFN19	CASHV4C2
CASHC	CASHDT03	CASHI	CASHV4C3
CASHC0	CASHDT04	CASHN	CASHV4C4
CASHC1	CASHDT05	CASHR	CASHV4D
CASHC2	CASHDT06	CASHR0	CASHV4UT
CASHC3	CASHDT07	CASHU	
CASHC4	CASHDT08	CASHUT	
CASHC5	CASHFN11	CASHUT01	

See “Callable Routines, Entry Points, and APIs” for a detailed description of APIs. For information about other routines, run XUPRROU (List Routines). This command prints a list of the CASH routines. This option is found on the XUPR-ROUTINE-TOOLS menu on the XUPROG (Programmer Options) menu, which is a sub-menu of the EVE option (Systems Manager menu). The following screen capture demonstrates the use of this command:

```
Select Systems Manager Menu Option: programmer
Options
Select Programmer Options Option: routine tools
Select Routine Tools Option: list routines
Routine Print
Want to start each routine on a new page: No// [ENTER]
Routine(s) ? > CASH*
```

The first line of each routine contains a brief description of the general function of the routine. Use the Kernel option XU FIRST LINE PRINT (First Line Routine Print) to print a list of just the first line of each CASH subset routine.

```
Select Systems Manager Menu Option: programmer Options
Select Programmer Options Option: routine tools
Select Routine Tools Option: First Line Routine Print
PRINTS FIRST LINES
Routine(s) ? > CASH*
```

6. Exported Options

There are no menu options delivered with the CASH tool.

7. Exported RPCs

The CASH tool has no exported remote procedure calls.

8. Other Software Elements

There are no print, sort, input or list templates in CASH 1.0. There are also no bulletins and no mail groups.

9. Callable Routines, Entry Points, and APIs

The following supported reference calls allow other packages to access CASH calls. A Developer Manual is also available, and it discusses the use of these APIs in greater detail.

9.1 File Discovery

Name: START^CASH

Description: Initiates the file discovery process for a specified file.

Calling Syntax Set RETURN=\$\$START^CASH(FILE,FLAGS,PACKAGE,ID,OWNER,.LIST)
Or
Do START^CASH(FILE,FLAGS,PACKAGE,ID,OWNER,.LIST)
Return Value 1 = Success, 0 = Failure

Input	Output
FILE (Required) The FileMan file # to be discovered. FLAGS (Optional) The discovery flags listed below. PACKAGE (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. ID (Optional) When the “I” flag is <i>not</i> passed in FLAGS, this input is ignored. When the “I” flag <i>is</i> passed in FLAGS, the ID input overrides the default ID string of “IEN” for each Class. OWNER (Optional) A valid Caché SQL user name or role. The Classes will be created with this user/role as the owner. If null, the default is _System. LIST (Optional) An array of fields to include in the mapped Class(es). The fields specified in the LIST parameter will be added to the generated Class(es), along with any required fields. All other fields will be ignored. See “The List Variable and Multiple or Pointer Fields” for more information about this parameter.	1 = Success 0 = Failure

Input	Output
<p>NOTE: If a value for 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>	

Discovery Flags

The following flags may be used with this API. Note that all flags are case-sensitive – for example, the flags F and f are not equivalent.

C	<p>Compile - Create and compile the Caché classes after the file(s) have been discovered.</p> <p>Important Note:</p> <p>If “C” is passed in, the CREATE^CASH call will be executed with all other parameters passed straight through. If “C” is not passed, all parameters other than FILE and all flags other than the “V” Flag are ignored by START^CASH!</p>
D	<p>Descriptions – Adds the full field description from the data dictionary to the Caché Property description. This data is not retrieved from file #15050.11—it is just copied directly from the data dictionary at compilation time.</p> <p>Adding the full description means that this text is viewable in the HTML Class documentation (generated by the DocBook class in Caché).</p>
E	<p>Expand - Pointers will generate a computed Property that expands the .01 field (by default) in the pointed-to file.</p> <p>This flag can be used in conjunction with the “P” flag.</p> <p>If you want to expand a field in the pointed-to file, other than the .01 default, generate the Class, and then use Caché to amend the PFIELD parameter of the computed Property and re-compile the Class.</p>
F	<p>Force - Forces the creation and compilation of a Class even if it already exists. The old version of the Class will be deleted. This flag should not be passed if users are running Caché queries against classes that will be deleted, as it will crash those queries.</p>
I	<p>Simple IDs – Class generation will use a simple ID rather than one generated from the Class name. The file IEN will be used, unless an alternative ID is passed (see the Input/Output table, above).</p>
L	<p>Loose Validation – This flag relaxes the constraints placed on Properties, so exporting data to a third-party SQL database should be easier. The required field flags will be ignored, and date/time datatypes will be created as varchar datatypes. You are less likely to get import errors when the FileMan data does not meet its own constraints.</p>

M	<p>Multiples - Class generation will include Sub-Classes for Multiple fields and create the appropriate parent-child relationships.</p> <p>If this flag is used without either the r or R flag, any second-level Multiple will be ignored (that is, the target file and its Multiples will be generated as Classes, but any Multiples inside the parent's multiples will not be generated).</p>
N	<p>Simple Names – Generation will use simpler names, in the format <i>[file name][file number]</i>.</p> <p>If flag M is also set, the sub-Classes generated for Multiples will be named without the prefix of their parent and grandparent names.</p>
P	<p>Pointers - Generate Classes for pointed-to files and create the appropriate foreign keys. This flag can be used with the “E” flag.</p>
Q	<p>SQL Only Compile – Only tables are compiled, not full classes.</p> <p>Caché Classes generally provide multiple types of access (e.g. SQL and Object). Using the “Q” flag restricts access to SQL only, so objects cannot be instantiated and classmethods cannot be called.</p> <p>This flag is useful if you know you only need SQL access, or want to restrict access to SQL-only.</p>
R	<p>Recursive – Generation of Multiples and Pointers will generate Sub-Classes and Classes for their Multiples and Pointers. Recursion will continue in this fashion until all levels of Multiples and Pointers have been exhausted.</p> <p>WARNING: Using “R” can generate lots of Classes! Ensure that you have adequate disk space available before using this flag. See the “Initial Disk Utilization Growth when 7400 FileMan Files are Mapped to Caché Classes” table in this document for an estimate of how much disk space might be required.</p> <p>If the R flag is passed, it will override the r flag.</p>
r	<p>Partially Recursive – Generation of Classes for Multiples, and Multiples inside of Multiples, will complete as with the R flag for full recursion. Generation of classes for Pointers will be completed through one iteration only – a Pointer in a parent file, a child Multiple or a Multiple's Multiple (etc.) will result in a generated Class for that Pointer; but, any Pointers and Multiples in the Pointed-to file will be ignored.</p> <p>If the R flag is passed, it will override the r flag.</p>

S	<p>SOAP Web Services - Create a new sister Class, inherited from %SOAP.WebService. The sister Class will contain SQL stored procedures that are generated for each index in the main Class (plus the default ByID procedure).</p> <p>These stored procedures are also exposed as web Methods. You can invoke a test Web page with the following URL:</p> <p>http://servername:1972/csp/namespace/package.classWS.cls</p> <p>e.g http://127.0.0.1:1972/csp/VistA/HDIS.StateWS.cls</p> <p>You can also view the WSDL Service Description by appending ?WSDL=1:</p> <p>http://servername:1972/csp/namespace/package.classWS.cls?WSDL=1</p> <p>e.g http://127.0.0.1:1972/csp/VistA/HDIS.StateWS.cls?WSDL=1</p> <p>Note: This functionality is supported in Caché Version 5.0.* and later only!</p>
V	<p>Verbose – This flag overrides the default Silent Mode of the CASH tool. The Caché compile messages will not be affected by this flag, they print to screen by default. This flag can be used with or without the C flag.</p>
W	<p>Web Page – Generated Classes will have the %CSP.Page Super Class. This flag can only be used with the X flag. If the X flag is not passed, this flag will be ignored.</p> <p>In the generated Classes, the OnPage() method will be overridden to display the output of the XMLDump() method.</p> <p>Note: This flag is supported with Caché version 5.0.* and later only! This flag will be ignored when the local Caché system is less than version 5.0.</p>
X	<p>XML – Generated classes will have the %XML.Adaptor super Class.</p> <p>Each Class will also have an XMLDump() method that exports the entire file in XML format. The XMLDump() method uses the inherited XMLExport() instance method.</p> <p>Note: This flag is supported with Caché version 5.0.* and later only! This flag will be ignored when the local Caché system is less than version 5.0.</p>

The List Variable and Multiple or Pointer Fields

To specify fields in Multiples or Pointers, use the relevant Flags (“M” or “P”) and add array nodes as follows. To add more than one Multiple or Pointer, type each one as a separate LIST parameter:

```
LIST (multiple_file#, field#)=""
LIST (pointer_file#, field#)=""
```

For example:

```
LIST(4,.01)=""      - Adds the .01 field, Name, from file 4, Institution.
LIST(4,.02)=""      - Adds the .02 field, State Pointer, from file 5, State.
LIST(5,.01)=""      - Adds the .01 field, Name, from file 5.
```


Input	Output
<p>will be ignored.</p> <p>See “The List Variable and Multiple or Pointer Fields” for more information about this parameter.</p> <p>NOTE: If a value for 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>	

Flags

The following flags may be used with this API. Note that all flags are case-sensitive – for example, the flags F and f are not equivalent.

D	<p>Descriptions – Adds the full field description from the data dictionary to the Caché Property description. This data is not retrieved from file #15050.11—it is just copied directly from the data dictionary at compilation time.</p> <p>Adding the full description means that this text is viewable in the HTML Class documentation (generated by the DocBook class in Caché).</p>
E	<p>Expand - Pointers will generate a computed Property that expands the .01 field (by default) in the pointed-to file.</p> <p>This flag can be used in conjunction with the “P” flag.</p> <p>If you want to expand a field in the pointed-to file, other than the .01 default, generate the Class, and then use Caché to amend the PFIELD parameter of the computed Property and re-compile the Class.</p>
F	<p>Force - Forces the creation and compilation of a Class even if it already exists. The old version of the Class will be deleted. This flag should not be passed if users are running Caché queries against classes that will be deleted, as it will crash those queries.</p>
I	<p>Simple IDs – Class generation will use a simple ID rather than one generated from the Class name. The file IEN will be used, unless an alternative ID is passed (see the Input/Output table, above).</p>
L	<p>Loose Validation – This flag relaxes the constraints placed on Properties, so exporting data to a third-party SQL database should be easier. The required field flags will be ignored, and date/time datatypes will be created as varchar datatypes. You are less likely to get import errors when the FileMan data does not meet its own constraints.</p>

M	<p>Multiples - Class generation will include Sub-Classes for Multiple fields and create the appropriate parent-child relationships.</p> <p>If this flag is used without either the r or R flag, any second-level Multiple will be ignored (that is, the target file and its Multiples will be generated as Classes, but any Multiples inside the parent's multiples will not be generated).</p>
N	<p>Simple Names – Generation will use simpler names, in the format <i>[file name][file number]</i>.</p> <p>If flag M is also set, the sub-Classes generated for Multiples will be named without the prefix of their parent and grandparent names.</p>
P	<p>Pointers - Generate Classes for pointed-to files and create the appropriate foreign keys. This flag can be used with the “E” flag.</p>
Q	<p>SQL Only Compile – Only tables are compiled, not full classes.</p> <p>Caché Classes generally provide multiple types of access (e.g. SQL and Object). Using the “Q” flag restricts access to SQL only, so objects cannot be instantiated and classmethods cannot be called.</p> <p>This flag is useful if you know you only need SQL access, or want to restrict access to SQL-only.</p>
R	<p>Recursive – Generation of Multiples and Pointers will generate Sub-Classes and Classes for their Multiples and Pointers. Recursion will continue in this fashion until all levels of Multiples and Pointers have been exhausted.</p> <p>WARNING: Using “R” can generate lots of Classes! Ensure that you have adequate disk space available before using this flag. See the “Initial Disk Utilization Growth when 7400 FileMan Files are Mapped to Caché Classes” table in this document for an estimate of how much disk space might be required.</p> <p>If the R flag is passed, it will override the r flag.</p>
r	<p>Partially Recursive – Generation of Classes for Multiples, and Multiples inside of Multiples, will complete as with the R flag for full recursion. Generation of classes for Pointers will be completed through one iteration only – a Pointer in a parent file, a child Multiple or a Multiple's Multiple (etc.) will result in a generated Class for that Pointer; but, any Pointers and Multiples in the Pointed-to file will be ignored.</p> <p>If the R flag is passed, it will override the r flag.</p>

S	<p>SOAP Web Services - Create a new sister Class, inherited from %SOAP.WebService. The sister Class will contain SQL stored procedures that are generated for each index in the main Class (plus the default ByID procedure).</p> <p>These stored procedures are also exposed as web Methods. You can invoke a test Web page with the following URL:</p> <p>http://servername:1972/csp/namespace/package.classWS.cls</p> <p>e.g http://127.0.0.1:1972/csp/VistA/HDIS.StateWS.cls</p> <p>You can also view the WSDL Service Description by appending ?WSDL=1:</p> <p>http://servername:1972/csp/namespace/package.classWS.cls?WSDL=1</p> <p>e.g http://127.0.0.1:1972/csp/VistA/HDIS.StateWS.cls?WSDL=1</p> <p>Note: This functionality is supported in Caché Version 5.0.* and later only!</p>
V	<p>Verbose – This flag overrides the default Silent Mode of the CASH tool. The Caché compile messages will not be affected by this flag, they print to screen by default. This flag can be used with or without the C flag.</p>
W	<p>Web Page – Generated Classes will have the %CSP.Page Super Class. This flag can only be used with the X flag. If the X flag is not passed, this flag will be ignored.</p> <p>In the generated Classes, the OnPage() method will be overridden to display the output of the XMLDump() method.</p> <p>Note: This flag is supported with Caché version 5.0.* and later only! This flag will be ignored when the local Caché system is less than version 5.0.</p>
X	<p>XML – Generated classes will have the %XML.Adaptor super Class.</p> <p>Each Class will also have an XMLDump() method that exports the entire file in XML format. The XMLDump() method uses the inherited XMLExport() instance method.</p> <p>Note: This flag is supported with Caché version 5.0.* and later only! This flag will be ignored when the local Caché system is less than version 5.0.</p>

The List Variable and Multiple or Pointer Fields

To specify fields in Multiples or Pointers, use the relevant Flags (“M” or “P”) and add array nodes as follows. To add more than one Multiple or Pointer, type each one as a separate LIST parameter:

```
LIST (multiple_file#, field#)=""
LIST (pointer_file#, field#)=""
```

For example:

```
LIST(4,.01)=""      - Adds the .01 field, Name, from file 4, Institution.
LIST(4,.02)=""      - Adds the .02 field, State Pointer, from file 5, State.
LIST(5,.01)=""      - Adds the .01 field, Name, from file 5.
```

LIST(5,3)=""	- Adds the 3 field, County, from file 5. This field is a multiple, file 5.01
LIST(5.01,.01)=""	- Adds the .01 field, County from file 5.01.

This example assumes that the M and P flags are also passed. The example produces three Classes (for files 4, 5, and 5.01). Each Class has one data Property and the relevant relationships and foreign keys. The map created from this example would include the institution name, and the state and county.

If the E flag is also passed, a computed field would also be generated for the Pointer in file 4.

9.3 Caché Class Creation for All FileMan Files

Name: ALL^CASH

Description: Invokes CREATE^CASH with parameters to discover all VistA files, and always passes the flags E, F, L, M, P, R, and V. This will cause Pointers and Multiples to be generated into Classes and sub-Classes, including all Multiple fields inside the Multiples and all files that are pointed to. Finally, the V flag prints the activity on the screen. Other flags may be optionally added.

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

Input	Output
<p>FLAGS (Optional)</p> <p>The discovery flags listed below.</p> <p>PACKAGE (Optional)</p> <p>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> <p>ID (Optional)</p> <p>When the "I" flag is <i>not</i> passed in FLAGS, this input is ignored. When the "I" flag <i>is</i> passed in FLAGS, the ID input overrides the default ID string of "IEN" for each Class.</p> <p>OWNER (Optional)</p> <p>A valid Caché SQL user name. The Classes will be created with this user as the owner. If null, the default is _System.</p>	

Flags

The following flags may be used with this API. Note that all flags are case-sensitive – for example, the flags F and f are not equivalent.

D	<p>Descriptions – Adds the full field description from the data dictionary to the Caché Property description. This data is not retrieved from file #15050.11—it is just copied directly from the data dicationary at compilation time.</p> <p>Adding the full description means that this text is viewable in the HTML Class documentation (generated by the DocBook class in Caché).</p>
I	<p>Simple IDs – Class generation will use a simple ID rather than one generated from the Class name. The file IEN will be used, unless an alternative ID is passed (see the Input/Output table, above).</p>
N	<p>Simple Names – Generation will use simpler names, in the format <i>[file name][file number]</i>.</p> <p>If flag M is also set, the sub-Classes generated for Multiples will be named without the prefix of their parent and grandparent names.</p>
Q	<p>SQL Only Compile – Only tables are compiled, not full classes.</p> <p>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.</p> <p>This flag is useful if you know you only need SQL access, or want to restrict access to SQL-only.</p>
S	<p>SOAP Web Services - Create a new sister Class, inherited from %SOAP.WebService. The sister Class will contain SQL stored procedures that are generated for each index in the main Class (plus the default ByID procedure).</p> <p>These stored procedures are also exposed as web Methods. You can invoke a test Web page with the following URL:</p> <p><code>http://servername:1972/csp/namespace/package.classWS.cls</code> e.g <code>http://127.0.0.1:1972/csp/VistA/HDIS.StateWS.cls</code></p> <p>You can also view the WSDL Service Description by appending ?WSDL=1: <code>http://servername:1972/csp/namespace/package.classWS.cls?WSDL=1</code> e.g <code>http://127.0.0.1:1972/csp/VistA/HDIS.StateWS.cls?WSDL=1</code></p> <p>Note: This functionality is supported in Caché Version 5.0.x only!</p>
W	<p>Web Page – Generated Classes will have the %CSP.Page Super Class. This flag can only be used with the X flag. If the X flag is not passed, this flag will be ignored.</p> <p>In the generated Classes, the OnPage() method will be overridden to display the output of the XMLDump() method.</p> <p>Note: This flag is supported with Caché version 5.0.* and later only! This flag will be ignored when the local Caché system is less than version 5.0.</p>

X	<p>XML – Generated classes will have the %XML.Adaptor super Class.</p> <p>Each Class will also have an XMLDump() method that exports the entire file in XML format. The XMLDump() method uses the inherited XMLExport() instance method.</p> <p>Note: This flag is supported with Caché version 5.0.* and later only! This flag will be ignored when the local Caché system is less than version 5.0.</p>
---	--

10. External Relations

10.1 Software Requirements

The following software should be installed prior to installing the CASH tool.

Software	Version	Patch Information
Caché	4.1 or higher	Standard installation.
VA FileMan	22.0	Fully patched.
Kernel	8.0	Fully patched.
Kernel Toolkit	7.3	Fully patched.
MailMan	8.0	Optional.

Table 5: Software Installation Requirements

10.2 DBA Approvals and Integration Agreements

There are no integration agreements.

11. Internal Relations

The CASH package has no internal relations and no input/output dependencies.

12. Software Product Security

This section discusses security changes and requirements introduced by CASH.

12.1 VHA Directives and Official Policies

Data Standardization is supported by VHA Directive 2005-044. A copy of this directive is publically available – see the “Reference Materials” section of this document for more information.

12.2 Legal Requirements

No legal requirements are introduced by this package.

12.3 Mail Groups and Alerts

No mail groups are created as part of the installation.

12.4 Remote Systems

The CASH package creates Caché Class objects on the local Caché system during file mapping.

Once Caché Classes have been created for the VistA files, data from the VistA files can be queried through protocols that are not supported by VistA. Most significantly, Caché supports ODBC and JDBC, where VistA does not. This change will make it possible for a user with access to the Caché Classes to retrieve data from the VistA files. Data transmitted is not encrypted during transmission.

Access to the Caché system should be configured, logged, and monitored according to best practices.

12.5 Menus and Options

No menu options are distributed with the CASH tool.

12.6 Security Keys and File Security

There are no security keys.

The table below indicates the security that CASH establishes for its files.

Number	Name	DD	RD	WR	DEL	LAYGO	AUDIT
15050.11	CASH FM CLASS MAP FILE	@	@	@	@	@	@
15050.12	CASH CUSTOM DATATYPES FILE	@	@	@	@	@	@
15050.13	CASH ERRORS FILE	@	@	@	@	@	@

Number	Name	DD	RD	WR	DEL	LAYGO	AUDIT
15050.14	CASH PARAMETER FILE	@	@	@	@	@	@
15050.19	CASH SQL RESERVED WORDS FILE	@	@	@	@	@	@

Table 6: Security Keys and File Security

13. Glossary

API	Application Programming Interface. This is the definition (calling conventions) by which one application can get services from another application.
Class (also Sub-Class)	<p>A Class is a collection of Properties that describes a set of objects and the Methods (or Behaviors) that can be applied to these objects.</p> <p>In this context, a class is a Caché database object, analogous to a table in a relational database. A Sub-Class is a child object, related to its the parent Class through a Foreign Key.</p> <p>In Caché, database objects can be defined as Classes, though these objects can also be exposed as Relational Tables (accessible via standard SQL), using Caché's Unified Data Architecture.</p>
Domain	A subset of medicine, a natural grouping of clinical acts (e.g., demographics, vital signs, laboratory, pharmacy)
DS	Data Standardization
GB	Gigabyte. Equal to 1024 megabytes.
HDI	Health Data and Informatics
HDR	Health Data Repository
IEN	Internal Entry Number, a number assigned to each entry in FileMan files.
JDBC	Java Database Connectivity. A standard API for Java programs to use to access databases.
Mapping	<p>Mappings are a set of relationships established between two formats for data.</p> <p>In this context, the Mapping Tool uses the metadata provided by the FileMan data dictionary to generate Caché Classes (and their implied Relational Tables) that can access the FileMan data. Each map is a template for the Caché Class or datatype generated from it.</p>
MB	Megabyte. Equal to 1024 kilobytes.
Method	<p>A procedure or sub-routine associated with a Class. Can also be referred to as a Behavior.</p> <p>In Caché, methods can be exposed as SQL Stored Procedures as well.</p>
Multiple	<p>A Multiple is a FileMan Field that is stored as an array. The field is hierarchically related to a parent file, allowing storage of zero, one or many values.</p> <p>A Multiple is equivalent to a child table in a relational database.</p>
ODBC	Open Database Connectivity. A standard API for accessing different database systems. An application can submit standard statements to ODBC, and ODBC will translate these to a language the database understands.

Package	For the purposes of this document, package always refers to a collection of Caché Classes.
Pointer	A Pointer is a FileMan field that links to another FileMan file. A Pointer is equivalent to a Foreign Key in a relational database.
Property	In Caché, a Property is a field or attribute in a class/table.
Standardization	The process of defining, creating, deploying, and maintaining a common terminology resource (i.e., content and services) to all current and future VHA applications.
SQL	Structured Query Language. An industry-standard language for creating, updating and querying relational database management systems. SQL is also an ISO and ANSI standard.
Terminology	Set of terms, definitions, relationships of a specialized subject area. The terms which are characterized by special reference within a discipline are called the 'terms' of the discipline, and collectively, they form the terminology, those which function in general reference over a variety of languages are simply 'words', and their totality 'the vocabulary' [Sager]. See also vocabulary.
Trigger	An action causing the automatic invocation of another action (such as a routine or procedure). A trigger “fires” when data is modified. Triggers are typically used to ensure that an update to one piece of data also updates all related data.
URL	Uniform Resource Locator - a Web page address on the World Wide Web.
VOID	VHA Unique Identifier - these are meaningless numbers that are automatically assigned to concepts, properties, and relationships in a terminology to facilitate their access and manipulation by computers.
Word Processing Field (also WP Field)	A Word Processing Field is a special multi-line field that can be used to store a large amount of text. WP Fields are analogous to a Memo field in a relational database management system. BMS. This data is stored in FileMan as an array (sub-file), very similar in structure to a Multiple. The main difference being that a WP Field has only one field in its sub-file, whereas a Multiple can have many.
XML	Extensible Markup Language – a text format used extensively for information exchange.