

VistA M Routine Analyzer

Release 2.00

Ray Group International

August 15, 2012

Abstract

This paper describes a set of M tags that can be used to analyze M routines and Fileman dictionaries in VistA installations for package to package dependencies, code quality, and package entry tags. In addition, a Java version is also provided and is used for testing purposes. Either version can serve as a starting point or a testbench for newly developed or existing software that is used for static M code analysis tools or automated refactoring.

Contents

1	Introduction	1
2	Repository	2
3	Code Walkthrough	2
4	How to use the code	13
5	Conclusions	15

1. Introduction

VistA code is organized in terms of packages. These packages depend on each other in a variety of ways. Understanding these dependencies is important both for discussions on iEHR architecture and VistA refactoring efforts. The M tags discussed in this paper can be used to generate various reports on these dependencies. In addition a Java based static M code analysis tool is provided as a testbench for the results that are generated by the M tags. Java M Routine analysis tool has potential to be used as a testbench for XINDEX or as a testbench for other tools, such as M to Java converters or as an automated refactoring tool or code beautifier. However, the latter stated possibilities are not explored in this paper.

The M code here is an improved and extended version of XINDEX utility which has been long used to validate new and edited routines for VistA. The code essentially parses all the routines in a VistA installation and stores relevant information in M Globals. The Global structures in this work

closely mimic what was used in XINDEX. In fact, whatever was being internally generated by XINDEX was left in place and new subscripts were added for various analysis needs. Global creation of tags are stand alone and can be used to generate reports of any kind. In this work we mainly provide text file reports that are somehow similar to XINDEX outputs.

The reports generated by VistA M Routine Analyzer are M entry point (tag) based. This is one of the key differences from the original XINDEX which generates routine based reports. Since M API's are specified in terms of tags this is more natural for Refactoring efforts.

Java code that is used as a test bench includes an M parser and parse tree generating classes. A visitor class is included to transverse the parse tree while all the report generating classes subclass the visitor. Java version reads VistA M files directly from OSEHRA Git repository.

The following are the information available in the reports.

1. For any tag:
 - a. Formal Parameters
 - b. Assumed Variables
 - c. Directly used Globals
 - d. Globals used through classic Fileman calls (partial)
 - e. File specified Fileman DBS calls
 - f. Number of Read commands
 - g. Number of Write commands
 - h. Number of Indirections used
 - i. Number of Xecute statements used
2. All entry tags in a package that are being called from other packages
3. All entry tags in other packages that are being called from a package
4. All tags in a package that are used in Options File
5. All tags in a package that are used in RPC calls.

2. Repository

The M code is being hosted in the following Git repository:

- <https://github.com/OSEHR/M-RoutineAnalyzer>

The Java code is being hosted in the following Git repository:

- <https://github.com/kthlhkeating/rqivistatools>

3. M Code Walkthrough

M entry point tags can be grouped into two groups: (1) those that generate data as Global entries and (2) those that generate reports. This distinction makes it possible for users to use the Globals to tailor their reporting reads instead of parsing the reports for data.

All the tags accept an optional first argument called GLB. This is where all the data is stored. If not specified, GLB defaults to "ZZRG".

Data Generating M Entry Point Tags

CRTBASE^ZZRGND14{GLB} – Create base data

Example

```
D CRTBASE^ZZRGND14("^ZZRG")
```

Description

This tag parses the code base and stores relevant information needed by other tags in the input Global. The parser here is essentially the one used in XINDEX with some fixes and modified so that additional information is stored. XINDEX based portion of the code uses ^UTILITY(\$J) Global to store the information and this information is then moved to GLB.

Since GT.M and Cache store routine information differently you will be asked to select routines. Routine selection is done by executing ^%ZOSF("RSEL"). The prompt is different depending on your host. On Cache installation please select the following to generate the Global for the whole codebase:

```
All Routines? No => Yes
```

On GT.M installations please select:

```
Routine: *
```

Note that no information is generated for "%" routines since these routines are typically environment dependent. Additionally, "%" routines are also ignored for all the analysis.

This tag kills the given Global first. If you want to update the Global for select routines run this tag with a different Global, kill the relevant routine nodes in the original, and do a merge. Since running this tag takes tens of minutes, this may become handy if you change a few routines and want to update the Global.

Generated Data

This tag creates three subscripts in the target Global: @GLB@(0), @GLB@(1), and @GLB@2.

@GLB@(0) and @GLB@(1) are what the original XINDEX generated and used in its reports. No intentional change has been made to these subscripts, however, due to our changes and fixes in the code, some entries will be different from what XINDEX generates. We do not currently do any comparison test to XINDEX reports. The following describes what is stored in these subscripts:

```
@GLB@(0)="N;ROU" where N is the number of routines
@GLB@(1,RTN,0,0)=Number of lines in the routine
@GLB@(1,RTN,0,LINE_NO,0)=Source code
@GLB@(1,RTN,"RSUM")=Routine check sum value
@GLB@(1,RTN,"COM",LINE_NO)=Line label and commands separated by <TAB>
@GLB@(1,RTN,"T",TAG)=No value, keys the tags in the routine
@GLB@(1,RTN,"E",0)=Number of errors in the routine
@GLB@(1,RTN,"E",ERROR_NO)=Error code, error location (tag+offset) and
error text separated by <TAB>
@GLB@(1,RTN,"G",GLOBAL,0)=Comma separated locations (tag+offset)
where GLOBAL is used
@GLB@(1,RTN,"L",LOCAL)=Flags (~=Newed,*=Initialized,!=Killed)
@GLB@(1,RTN,"L",LOCAL,0)=Comma separated locations (tag+offset)where
LOCAL is used appended by flags (~=Newed,*=Initialized,!=Killed)
```

Latest version available at the [OSEHRA Journal](http://hdl.handle.net/10909/2) [<http://hdl.handle.net/10909/2>]

Distributed under [Creative Commons Attribution License](#)

@GLB@(1,RTN,"I",TAG,0)=Comma separated locations (tag+offset) where internal tags are called

@GLB@(1,RTN,"N","^(",0)=Comma separated locations (tag+offset) where a naked global is used

@GLB@(1,RTN,"X",ROUTINE TAG,0)=Comma separated locations (tag+offset) where external tags are called.

The subscript @GLB@(2) includes either additional information that is not available in @GLB@(1) or indexes the same information in an ordered way. For example, for assumed variable determination whether the order of a local is newed and set or read is important. The following describes what is stored in this subscript:

@GLB@(2,RTN,TAG,"L")=Level of the line where tag is

@GLB@(2,RTN,TAG,"N")=Next tag in line number order

@GLB@(2,RTN,TAG,"P")=Previous tag in line number order

@GLB@(2,RTN,TAG,LINE_NO)=Number of additional data subscripts

@GLB@(2,RTN,TAG,LINE_NO,I)=Additional data (described below)

@GLB@(2,RTN,TAG,LINE_NO,I,"LVL")=Level of the line

@GLB@(2,RTN,TAG,LINE_NO,"PRS")=Number of additional data subscripts

@GLB@(2,RTN,TAG,LINE_NO,"PRS",I)= Additional data (described below)

Here LINE_NO is with respect to the tag. Numbered additional data can be as follows:

- "CMD"_\$C(9)_CMD: CMD is the long form of M commands such as DO or SET
- "L"_\$C(9)_NAME_\$C(9)_FLAG_\$C(9)_P3: Local information. P3 is not used. FLAG can be as follows
 - o ~: Newed
 - o *: Set
 - o !: Killed
 - o n: Formal parameter
 - o p;n: Passed as reference in the nth parameter
 - o <empty>: Read
- "@_\$C(9)_NAME_\$C(9)_P2: Indirection. P2 is not used.
- "G"_\$C(9)_NAME_\$C(9)_FLAG_\$C(9)_P3: Global information. P3 is not used. FLAG is similar to locals for set, killed and read.
- "Q"_\$C(9,9,9): Marks the termination of a entry point tag. A termination point is a QUIT command such that
 - o It is not on a line of level 2 or bigger
 - o It is not a termination for a FOR loop
 - o It is not under IF or ELSE command
 - o It does not have a postcondition
- "X"_\$C(9)_RTN TAG_\$C(9)_TYPE_\$C(9)_P3: External DO or GOTO command branching. TYPE can be D0, D1, G0 and G1 corresponds to DO, DO with postcondition, GOTO and GOTO with postcondition. P3 is 1 or 0 depending on the first 3 conditions of "Q" above.
- "I"_\$C(9)_TAG_\$C(9)_TYPE_\$C(9)_P3: Internal DO or GOTO command branching. TYPE and O3 are similar to "X" above.

"PRS" additional data can be as follows

- "S"_\$C(9)_LOCALNAME_\$C(9)_LOCALVALUE

- "S"_\$C(9)_LOCALNAME1,LOCALNAME2_\$C(9)_LOCALVALUE
- "D"_\$C(9)_ROUTINE TAG

READPKGS^ZZRGND19(GLB,CLEAN,PKGSPATH) – Read Packages.csv

Example

```
D READPKGS^ZZRGND19("^TARGET",1,"C:\Directory\)
```

Description

This tag reads Packages.csv which includes VistA package namespace information. The information is stored in @GLB@(10). This information is read from this file instead of VistA Packages file because this file contains information that is more up to date.

If CLEAN is true, the @GLB@(10) is first killed. PKGSPATH is the directory where the Packages.csv is located. This parameter should end with the directory separator that the environment uses.

Each package is identified with a default namespace (first one on the list) which are used in other tags in this work as an input to identify a package. The following describes what is stored in @GLB@(10)

Generated Data

```
@GLB@(10,NAMESPACE)=PACKAGENAME
@GLB@(10,NAMESPACE,"D")=DEFAULTNAMESPACE
```

RDOWNER^ZZRGND19(GLB,CLEAN,OWNPATH) – Read Ownership.csv

Example

```
D RDOWNER^ZZRGND19("^TARGET",1,"C:\Directory\)
```

Description

This tag reads Ownership.csv which includes VistA Global ownership information. OWNPATH is where this file is located. The information is stored in @GLB@(11). This information is read from this file instead of VistA Packages file because this file information is more up to date.

If CLEAN is true, the Global is first killed.

Generated Data

```
@GLB@(11,GLOBAL)=NAMESPACE_"^"_PKGNAME_"^"_FILENAME_"^"_FILENUMBER
@GLB@(11,GLOBAL,SUBSCRIPT)=Same as above
```

NDXFI^ZZRGND14(GLB) – Index fanin data

Example

```
D NDXFI^ZZRGND14("^TARGET")
```

Description

This tag indexes call between packages. Indexed data is stored in @GLB@(4), @GLB@(8) and @GLB@(9). This information is then used in other tags for reporting. Note that \$TEXT intrinsic function is not considered a call between packages and is not stored. This differs from XINDEX output.

Generated Data

```
@GLB@ (4, FANINPKG) = Number of packages calling
@GLB@ (4, FANINPKG, FANOUTPKG) = ""
@GLB@ (8, FANOUTPKG, FANINPKG, FANINRTN, FANINTAG) = ""
@GLB@ (9, FANINPKG, FANINRTN, FANINTAG, FANOUTPKG, TAG^FANOUTRTN) = ""
```

NDXFMAN^ZZRGND17(GLB) - Index Fileman calls

Example

```
D NDXFMAN^ZZRGND17("^TARGET")
```

Description

This tag indexes information for Fileman. Two kinds of Fileman call information are considered: (1) classic fileman calls and (2) DBS calls.

Classic Fileman calls are found from variables DIC, DIE, DIK. If these variables are set to a constant string that is a Global (starts with "^"), then it is assumed that there is a Fileman call in the routine where the variable is assigned.

All calls to a Fileman tag (found by the namespace of the routine) are assumed to be a DBS call if the first argument is a constant and is a file number (19, 19.5, 0.55, etc.).

Generated Data

```
@GLB@ (7.5, PKG, RTN, "FMG", FILEMANGLOBAL) = ""
@GLB@ (7.5, PKG, RTN, "FMGC", FILEMANCALL) = ""
```

MAIN^ZZRGND14(GLB,PKGSPATH) - Main data generator

Example

```
D MAIN^ZZRGND14("^TARGET","C:\Directory\)
```

Description

Convenience tag that calls CRTBASE^ZZRGND14, NDXFI^ZZRGND14, NDXFMAN^ZZRGND17 and READPKGS^ZZRGND19. See those tags.

UPDEINFO^ZZRGND22(RESULT,GLB,ENTRIES,RST,FLAG) - Update Entry Information

Example

```
D UPDEINFO^ZZRGND22(.RESULT,"^TARGET",.ENTRIES,1)
```

Description

This tag finds various information regarding a set of entry point tags specified in ENTRIES and stores the information in @GLB@(7) for access. It assumes MAIN^ZZRGND14 has already been called. Some of the information, such as assumed variables and use of indirections, indicate code quality/complexity while others, such as directly accessed Globals or Fileman calls, are indicative of dependence to other packages.

ENTRIES is a one based array where ENTRIES itself holds the number of entries and ENTRIES(I) holds the entry points of the form TAG^ROUTINE. RST can be set to true to kill @GLB@(7). If there are overall errors such as no numeric subscripts in ENTRIES, RESULT is set to 0^ERROR. Otherwise RESULT is set to 1 and RESULT(I) is set to 1^PKG^RTN_TAG for

success and 0^ERROR for failure.

The end of an entry point tag is found by identifying the first QUIT command that:

1. Does not have postconditional
2. Is not under FOR block
3. Is not under IF/ELSE block
4. Is not on a line that has level larger than 1
5. First GOTO command that satisfies both 1 and 3 above

If no such QUIT or GOTO command exists then the entry tag ends at the end of the routine.

The information is recursively additive. For example, assumed variables for an entry point tag includes not only the assumed variables in the tag but also the assumed variables in all the external and internal tags that are called. FLAG can be used to control this behavior. It can take values 0, 1, 2, and 3.

- 0: Include information from all the tags that are called in the code flow
- 1: Exclude information from Fileman and Kernel tags
- 2: Exclude information from the tags in other packages
- 3: Exclude information from external tags (other routines)

Generated Data

```
@GLB@ (7, PKG, RTN, TAG, INFOID) = INFOVALUE
```

INFOID and INFOVALUE can be as follows:

```
..., "F") = Number of formal parameters.
..., "F", I) = Formal parameter
..., "A", ASSUMEDLOCAL) = ""
..., "G", DIRECTLYACCESEDGLOBAL) = ""
..., "@" = Number of indirection used
..., "X") = Number of executable commands used
..., "W") = Number of write commands used
..., "R") = Number of read commands used
..., "FMG", FILEMANGLOBAL) = ""
..., "FMGC", FILEMANCALL) = ""
```

NDXOPT^ZZRGND14(GLB) – Index Option file entry points

Example

```
D NDXOPT^ZZRGND14("^TARGET")
```

Description

This tag finds Option file (19) entry points that are used in roll-and-scroll interface and stores them in @GLB@(13).

Generated Data

```
@GLB@ (13, PKG, OPTIONNAME, RTN, TAG) = ""
```

NDXRPC^ZZRGND14(GLB) – Index RPC entry points

Example

```
D NDXRPC^ZZRGND14(^TARGET)
```

Description

This tag finds RPX entry points that are used in roll-and-scroll interface and stores them in @GLB@(14) and @GLB@(14.5).

Output

```
@GLB@ (14,PKG,RPCNAME,RTN,TAG)=""
@GLB@ (14.5,PKG,RTN,TAG)=""
```

Report Generating M Entry Point Tags

These entry points generate reports based on generated Globals that are described in the previous section. They also serve as examples for programmers who might want to have their own custom reports.

Each report accepts GLB as the first parameter. GLB is the Global where the information is stored and the report is based. If not specified, it defaults to ^ZZRG. Also an output file can be specified by FILEPATH and FILENAME parameters. If neither of them is specified then the report is written to the terminal. Most reports can be restricted to a particular package by the parameter PKG. Unless otherwise specified, this is an optional parameter and if not specified the report is written for all the packages.

REPORTFO^ZZRGND13(GLB,PKG,FILEPATH,FILENAME) – Report fanouts

Examples

```
D REPORTFO^ZZRGND13(^TARGET,"GMPL")
```

```
D REPORTFO^ZZRGND13(^TARGET,,"C:\TargetDir\","report.txt")
```

Description

This tag assumes MAIN^ZZRGND14 has been run to produce @GLB@(4), @GLB@(8), and @GLB@(10). It writes a fanout report for a specified package or for all packages.

REPORTFI^ZZRGND13(GLB,PKG,FILEPATH,FILENAME) – Report fanins

Examples

```
D REPORTFI^ZZRGND13(^TARGET,"GMPL")
```

```
D REPORTFI^ZZRGND13(^TARGET,,"C:\TargetDir\","report.txt")
```

Description

This tag assumes MAIN^ZZRGND14 has been run to produce @GLB@(4), @GLB@(9), and @GLB@(10). It writes a Fanin report for a specified package or for all packages.

OPTRTNS^ZZRGND24(GLB,PKG,FILEPATH,FILENAME) – Report option routines

Examples

```
D OPTRTNS^ZZRGND24("^TARGET","GMPL")
```

```
D OPTRTNS^ZZRGND24("^TARGET",,"C:\TargetDir\","report.txt")
```

Description

This tag assumes MAIN^ZZRGND14 has been run to produce @GLB@(10). It writes a report that contains all the entry points for roll-and-scroll interface.

RPCRTNS^ZZRGND24(GLB,PKG,FILEPATH,FILENAME) – Report option routines

Examples

```
D RPCRTNS^ZZRGND24("^TARGET","GMPL")
```

```
D RPCRTNS^ZZRGND24("^TARGET",,"C:\TargetDir\","report.txt")
```

Description

This tag assumes MAIN^ZZRGND14 has been run to produce @GLB@(10). It writes a report that contains all the entry points for Remote Procedure Calls for a specified package or for all packages

USES^ZZRGND15(GLB,PKG,OWNPATH,FILEPATH,FILENAME) – Report Globals package uses

Examples

```
D USES^ZZRGND24("^TARGET","GMPL","C:\OwnDir\")
```

```
D USES^ZZRGND24("^TARGET","GMPL",,"C:\OwnDir\","C:\TargetDir\","report.txt")
```

Description

This tag assumes MAIN^ZZRGND14 has been run to produce @GLB@(1) and @GLB@(10). It writes a report that contains the Globals that the specified package directly uses as well as the owners of the package. OWNPATH is a required parameter to specify the location of the ownership.csv file. PKG is also required in this tag .

USED^ZZRGND15(GLB,PKG,OWNPATH,FILEPATH,FILENAME) – Report package's used glbs

Examples

```
D USED^ZZRGND24("^TARGET","GMPL","C:\OwnDir\")
```

```
D USED^ZZRGND24("^TARGET","GMPL",,"C:\OwnDir\","C:\TargetDir\","report.txt")
```

Description

This tag assumes MAIN^ZZRGND14 has been run to produce @GLB@(1) and @GLB@(10). It writes a report that contains the Globals that belong to the specified package and is directly used by other packages. OWNPATH is a required parameter to specify the location of the ownership.csv file. PKG is also required in this tag.

RFMCALLS^{^ZZRGND17}(GLB,FILEPATH,FILENAME) – Report Fileman Calls

Examples

```
D RFMCALLS^ZZRGND17("^TARGET")
```

```
D RFMCALLS^ZZRGND17("^TARGET","C:\TargetDir\","report.txt")
```

Description

This tag assumes MAIN^{^ZZRGND14} has been run to produce @GLB@(2) and @GLB@(10). It writes a report that contains the Globals that are used in classic Fileman calls and Fileman DBS calls such that the first argument is a file number.

WRESINFO^{^ZZRGND23}(GLB,ENTRIES,FILEPATH,FILENAME,FLAG) – Write Info on Entry Points

Examples

```
N ENTRIES S ENTRIES=2,ENTRIES(1)="TAG1^RTN1",ENTRIES(2)="TAG2^RTN2"
```

```
D WRESINFO^ZZRGND23("^TARGET",.ENTRIES, ",","C:\TargetDir\","report.txt",0)
```

Description

This tag is a user interface for UPDEINFO^{^ZZRGND22} and writes a report based on the content of @GLB@(7). The content of the Global is described in that tag as well as the parameters ENTRIES and FLAG. In the report the information is organized as follows

FORMAL: List of formal parameters
ASSUMED: List of assumed locals
GLBS: List of directly used Globals
READ: Number of Read commands
WRITE: Number of Write commands
EXEC: Number of Write commands
IND: Number of Write commands
FMGLBS: Globals used in classic fileman calls
FMCALLS: Fileman DBS calls

WRESINFP^{^ZZRGND23}(GLB,PATTERN,FILEPATH,FILENAME,FLAG)

Examples

```
D WRESINFO^ZZRGND23("^TARGET","1""GMPL""UN, ", "C:\TargetDir\","report.txt",0)
```

Description

This is similar to WRESINFO^{^ZZRGND23} except that the entries are specified using routine name patterns with PATTERN. The entries are then all the tags in the routine. Otherwise the output is similar to WRESINFO^{^ZZRGND23}.

4. Validation Code Walkthrough

All reports that are generated by M entry points are validated through a completely independent Java validation framework which generate the very same reports. Java validation framework is an M parser that generates M parse trees for M routines for which there is an available Visitor class. All the reports are generated by

Latest version available at the [OSEHRA Journal](http://hdl.handle.net/10909/2) [<http://hdl.handle.net/10909/2>]

Distributed under [Creative Commons Attribution License](#)

subclasses of the Visitor class. The framework operates on the VistA-FOIA Git repository directly without need for any M environment.

The framework uses Java command line arguments to generate the same reports that M entry point tags generate. The main class is `com.raygroupintl.vista.tools.MRoutineAnalyzer` which in turn uses various static inner classes in `com.raygroupintl.vista.tools.RunTypes` to handle command line arguments to generate reports. All reports expect that the environment variable `VistA-FOIA` is defined and points to the repository location.

Command Line Arguments For Reports

What follows are the command line arguments that can be used to generate the report that specified M entry point tags perform. For all the cases package specification is optional and if it is not specified the report is written for all packages. This is behavior similar to M tags. The code `<package_prefix>` can be used instead of `<package_name>`.

1. `fanout -o <output_file_path> -p <package_name>: REPORTFO^ZZRGND13`
2. `fanin -o <output_file_path> -p <package_name>: REPORTFI^ZZRGND13`
3. `option -o <output_file_path> -p <package_name>: OPTRTNS^ZZRGND24`
4. `rpc -o <output_file_path> -p <package_name>: RPCRTNS^ZZRGND24`
5. `usesglb -o <output_file_path> -p <package_name> -ownf <ownership_file_path>: USES^ZZRGND15`
6. `usedglb -o <output_file_path> -p <package_name> -ownf <ownership_file_path>: USED^ZZRGND15`
7. `filemancall -o <output_file_path>: RFMCALLS^ZZRGND17`
8. `parsetreesave -ptg <output_directory_path>: This is used in to save object form of the parse trees for each routine.`
9. `entry -o <output_file_path> -p <package_name> -r <regexp_routinename_0> -r <regexp_routinename_1>: This is used to write a list of entry points for the regular expression specified routines. You can specify multiple regular expressions.`
10. `entryinfo -o <output_file_path> -i <entry_points> -ptd <parse_tree_save_dir> -f <option_flags>: WRESINFO^ZZRGND23, WREAINFP^ZZRGND23`

For `entryinfo` command line options you can omit the `-ptd` option but the generation is typically slower. If you loaded additional M routines to the environment other than VistA-FOIA you can use `-md <m_routine_directory>` to specify the location of the additional M directories. Additional M routines are put into respective packages based on their names.

Package Overview

The following are the description of some of the main java packages used in the validation tool. There are JUnit tests for most of the packages in the repository which can serve as examples.

- *com.raygroupintl.parser*: This package implements a generic parser that uses a custom BNF like description.
- *com.raygroupintl.m.token*: This builds on *com.raygroupintl.parser* to implement M grammar. The grammar is specified in *com.raygroupintl.m.token.MTFSupply* and includes both a Cache version and a 1995 M Standard version. M Tokens are implemented as classes.
- *com.raygroupintl.m.parsetree*: This includes classes to describe the parse tree. The parse tree itself generated by classes in *com.raygroupintl.m.token* package classes and in particular *com.raygroupintl.m.token.MRoutine*. There is a visitor class *com.raygroupintl.m.parsetree.Visitor* which implements a Visitor class. All the reports are

based on subclasses of the Visitor class.

- *com.raygroupintl.m.parsetree.visitor*: Various visitor classes that are used in reporting.
- *com.raygroupintl.vista.repository*: Loads and maintains package information from Packages.csv.
- *com.raygroupintl.vista.repository.visitor*: This includes visitor classes to loop through the packages and package routines to generate reports.
- *com.raygroupintl.vista.tools*: This where the main class, MRoutineAnalyzer, is and also includes command line argument handlers.

5. How to use the code

All M tags can be run either from a Cache terminal or GT.M terminal. Dependency of tags on each other are described in the previous sections. A full example run of all reports, mainly concentrated on Problem List and Scheduling, is provided in GENRALL^ZRGND24. Note that both Ownership.csv and Packages.csv is assumed to be in C:\Sandbox for this run and all the outputs are written to C:\Sandbox as well. The results of this run are included in our directory in the repository.

Java files are provided in the repository as an Eclipse project. It is recommended that the validation tool is run right from Eclipse by specifying command line options as described in the previous section and VistA-FOIA environment variable. You can alternatively create a jar file from Eclipse and run the jar file using the specified command line arguments.

A composite command line option “mratb <sandbox_path> <sandbox_path_with_prefix>” is available. You can generate the same reports that are generated by GENRALL^ZRGND24 using “mratb C:\Sandbox C:\Sandbox\m_”. and two sets can be compared for validation. This is how we validate our results.

6. Conclusions

This paper presented a set of utility M entry point tags that does static code analysis and can be used during VistA code refactoring efforts. They are useful in the code analysis phase to find the dependencies between packages so we can get a better idea of what has to be done to make the code more modular.

Available information from the tags also include direct Global access, assumed local variables, number of execute, read, write statements, indirection usages for entry points, and provides information on code quality.

Finally a Java based equivalent for the tags is provided and is used for testing purposes.