OSEHRA

Open Source Electronic Health Record Alliance

Response to RFI VA118-17-N-1781

VistA Standardization, Virtualization, Consolidation, and Security Remediation

December 12, 2016

Table of Contents

Executive Summary

The U.S. Department of Veterans Affairs has been at the forefront of conceptualizing, developing and adopting a variety of innovative services and technologies related to care delivery for many decades. VistA remains a symbol of VA innovation and leadership but, like any major automated system, must keep pace with evolving user requirements and technologies. This RFI enunciates a set of major improvements that will ensure VistA remains a state of the art enabler for the delivery of care to our Veterans.

The August 2016 release of Executive Memorandum M-16-21, *Federal Source Code Policy: Achieving Efficiency, Transparency, and Innovation through Reusable and Open Source Software* will have a profound impact on future code development. The tasks described in this RFI provide an opportunity not only to demonstrate compliance with M-16-21, but also to benefit from it. The new Federal open source policy aims to achieve high efficiency of code reuse by the entire community of government agencies and the private sector. VA and the open source community have had significant successes in open source innovation that include the ongoing upgrade to Fileman, a highly successful collaborative Immunization project, a nascent code intake process, and multiple open community engagements. The recent launch of the follow-on eHMP effort as an open and agile project is one of many indications that VA has embraced open, agile innovation as an organizational strategy in this new era. This RFI response is informed by five years of community building and open source operation. While much of our experience is centered on the electronic health record component, the lessons learned are equally applicable at the platform level. Thus the OSEHRA Community is offering suggestions that will not only enhance the probability of a successful, sustainable implementation, but also continue VA's leadership in complying with Federal open source policy. For example:

- VA should continue to embrace open source, open architecture, open standards, and *open collaboration* in VistA development. Continued incremental changes are needed in its cultural and contracting approach to accommodate and encourage community participation. Such changes will not only result in a higher quality system for VA, but also create a wider support community to enhance sustainability. A collaborative approach will also enhance VA's ability to extend services and capabilities into the community beyond VHA. *VA should recognize that VistA is a national asset, and plan from inception to leverage the capability of the entire VistA community.*

- *The eHMP program currently in progress should become the centerpiece of the proposed evolutionary strategy*, and serve as a model for future open, agile development and integration. The current eHMP project should be modified to conduct at least part of the development in open collaboration with the open source community. The eventual replacement of CPRS will be critically dependent upon input from, and acceptance by, clinicians across the country.

- Leveraging open source requires more than simply distributing code and using a few open source products. *It involves systemic changes throughout the full life cycle of products and services.* Open source should not be an afterthought. Open collaboration, community input, and public-private partnership in sustainment should be "baked into" the program plan to reap the full benefits of open source.

Since its creation by VA in 2011, OSEHRA has accumulated unique expertise in open source operation as a successful public-private partnership. The established infrastructure of the OSEHRA Community can serve as a highly efficient ecosystem for such partnerships, and we hope to facilitate public-private partnership in pursuit of the goals outlined in this RFI.

1 OSEHRA and Open Source

1.1 The Open Source Electronic Health Record Alliance

In 2011, VA made a strategic decision to embrace public-private partnership as a means to accelerate innovation and reduce life cycle maintenance costs. The initial focus of this strategy was the electronic health record, and VA was particularly interested in engaging the VistA user community to access innovation and expertise in a more interactive manner rather than was possible through rigid contractual processes. The establishment of an open source community around VistA created the opportunity to go beyond simple distribution of the code for reuse. It allowed the community to contribute new code, bug fixes, enhancements, and other open source code management tools.

To establish, manage, and develop this open source community, VA established an independent nonprofit organization, the Open Source Electronic Health Record Alliance (OSEHRA). OSEHRA was chartered as a 501(c)(6) membership-based organization to build and support an open source ecosystem that would enable public-private collaboration based upon open source best practices that have evolved in the global IT community over the past 20 years.

VA, together with the OSEHRA community, has addressed a variety of challenges in implementing an open source approach. Initiating real collaboration between Government and the Community has posed cultural challenges, precipitated changes in internal development processes, and raised contractual issues when paid contractors are part of the project. VA has systematically addressed these and other emerging issues.

1.2 New Federal Policy on Reuse and Open Source

In August 2016, the Federal Government established a policy on open source through Memorandum M-16-21. Entitled *Federal Source Code Policy: Achieving Efficiency, Transparency, and Innovation through Reusable and Open Source Software*, the policy will bring about wide adoption of open source throughout the Federal government. The policy is very consistent with the ongoing open source innovations within VA, particularly the current efforts in community engagement. In fact, VA's experience and expertise offer an excellent framework for implementation and achievement of the reuse and community engagement goals of the new policy.

Some observations on this policy are relevant here. First, we believe the overarching goal of the policy is to make reuse of software efficient and transparent. The policy adopts open source as a technological and business process to enable code reuse. *Second, the policy specifically recommends community engagement and open development.* Finally, the policy is designed to ensure that open source is an integral part of agency policy, rather than an afterthought. OSEHRA has been working in all three of these areas with VA for years. Based upon our collective experience we have identified key challenges that VA must overcome, as well as suggestions to improve efficiency moving forward.

Efficient reuse of code requires a new way of thinking about the application of open source. Open source should not be an afterthought. Retroactive open source implementation makes the entire process expensive and inefficient for all users. VA's attempt to embed open source and

agile processes from the beginning of the current eHMP development effort is a strong indication that the lessons we all learned collectively are being applied to improve efficiency.

Compared with other agencies with no experience in open source, VA is years ahead. We believe our five-year experience in open source has laid an invaluable, strong foundation for the success of ODHP.

1.3 Our Approach for the OSEHRA Response

OSEHRA does not develop software or integrate systems. Instead, we work to build and support a community of developers, integrators, clinicians, and other interested parties to promote the concepts of open source and innovative collaboration. Our response to this RFI is thus based upon the experience and advice of our membership, and OSEHRA's direct experience as VA's open source facilitator. We held a Community Call on Monday, November 28, 2016 to encourage our members to provide direct responses keyed to the requirements in RFI. OSEHRA's response is offered from the perspective of an open source facilitator rather than a potential contractor, and focuses on areas where we believe the VA's open source strategy will be critical for success.

2 Open Source Perspective on Key RFI Requirements

2.1 Application, Data, and Business Rule Layers

The enhancements described in tasks iii, iv, and v include upgrades that OSEHRA has advocated for some time. Collectively, they have the potential to transform Vista into a true next-generation EHR. However, the skill set required to successfully implement a transformation of this magnitude does not exist in any contractor or contractor team. It is spread across an entire community of VistA developers and implementors. We believe that an open, transparent approach to the development described in this tasks is the only way to ensure success.

When refactoring VistA, it will also be important to develop in compliance with VA's Standards and Conventions (SAC). *Unfortunately, the SAC has not been updated to include critical improvements that will enhance the usability and maintainability of MUMPS code.* In December 2014, as part of our VA Gold Disk Support Contract, OSEHRA produced a set of recommendations to update the SAC. This deliverable (provided here as Appendix A), was developed by incorporating inputs from the entire OSEHRA community, and contains critical recommendations such as lengthening MUMPS routine names for better inherent documentation and ease of understanding/maintenance. *We strongly recommend that the recommendations in this document be implemented, and that a revised SAC be issued as part of any procurement.*

In pursing these tasks, particularly the refactoring, we urge VA to recognize that VistA is a national resource. Using an open, collaborative approach will not only result in better code for VA, but will also allow the thousands of external VistA users to keep pace with progress and benefit from VA's leadership and enhancements.

2.2 Security Layer

VA has made security its highest priority during the past few years, and that priority has produced credible results in several areas. The full breadth of security issues, including confidentiality, integrity, and availability, far exceeds the four items outlined in the RFI. For example, host-level security, local network topology, and device security will be critical components of any successful security approach.

There are several internal VA efforts that have significant potential to improve security posture and policy. For example, OI&T has recently conducted a series of open webinars with major medical device manufacturers, and instituted a Cooperative Research and Development Agreement (CRADA) with Underwriters Laboratories (UL) for current and emerging medical device cybersecurity standards and certification approaches. One result of this collaborative initiative is an excellent paper entitled *VA Enterprise Design Patterns: Privacy and Security Medical Device Security*[1]. The concepts and recommendations in this design pattern should be incorporated into any security efforts.

[1] Version 0.5 released November 2016.

Another internal effort that merits attention is the VistA Security Technical Working Group (TWG), convened under OSEHRA's *Open Source Technical and Community Engagement Support for VA VistA* contract. This is another collaborative effort, and involves a number of outside entities such as the Electronic Healthcare Network Accreditation Commission (EHNAC), commercial VistA implementer DSS, Inc., the National Health Information Sharing and Analysis Center (NH-ISAC), and multiple contractors with VistA implementation and network security expertise. The first product of this TWG will be a white paper on security a VistA instance. This paper, scheduled for release in early 2017, will cover a wide range of relevant topics, from VistA application security to device security and underlying network topography. We urge VA to ensure that the results of current collaborative efforts are utilized in securing VistA, and that any contract awards specify the continuation of open, collaborative, methods in addressing VA's security requirements.

3 Tools and Facilities for Implementation

3.1 Open Source Ecosystem

We would like to offer a comprehensive open source ecosystem that VA sponsors at OSEHRA. Open source is not a random collection of free software of unknown pedigree. Successful open source operations require a highly performing ecosystem of community, software and governance. OSEHRA's ecosystem has evolved to conduct open source operations in support of VistA Evolution. Each of these components will have critically important roles in making standardization, virtualization, consolidation and security remediation a lasting success.

3.2 Community

A community is the most important asset in open source activities, and the most important source of expertise, feedback, and contributions. The OSEHRA Community is a robust group of large and small businesses, nonprofits, academic institutions, government representatives, developers, and clinicians from around the world. Although it is incredibly diverse, OSEHRA has been able to engage its members to collaborate on joint projects and respond to challenges faced by VA. This is a great achievement considering that many community members are also industry competitors, and are geographically separated.

The OSEHRA Community's progress was highlighted in the IBM-commissioned Innovation Series Report titled *Making Open Innovation Ecosystems Work: Case Studies in Healthcare*. Published in 2015, this scholarly report examined VA's effort in building a "new ecosystem around its VistA in order to better facilitate the flow of innovations practices and processes between the VA and external agencies and private firms." After a detailed look at the OSEHRA Community, the authors noted that "OSEHRA has attempted to make all decisions transparent and open to the entire ecosystem, to allow everyone to see how any such decisions are made."

3.3 Working Groups

Most of the work of OSEHRA is done through organized working groups. For most groups, all interested community members are encouraged to participate. Often members of VA take a very active role in these activities, thus making government-private sector communication transparent, unambiguous and efficient. The following is a list of current working groups:

Group Name	Type
Architecture	Work Group
Certification	Work Group
Code Convergence	Work Group
Development Tools	Discussion Group
Immunization	Open Source Project Group
Johns Hopkins Oncology Clinical Information System (OCIS)	Discussion Group
Meaningful Use	Open Source Project Group
OSP Candidates for VA VistA Intake	Discussion Group
popHealth Developer	Work Group
popHealth Steering	Work Group
popHealth User	Work Group
Reminders VA	Work Group
VistA js	Work Group
VistA Security TWG	Work Group (Private)
Visualization	Open Source Project Group

OSEHRA's measured approach to engagement has enabled its community to effectively meet the needs of VA. When VA developed an Immunization Management Information System, a non-traditional method was used when VA approached OSEHRA's open source community rather than utilizing traditional software acquisition based on requirement documents. Community leaders organized an OSEHRA work group comprised of open source product developers, immunization experts from Indian Health Service, and VistA experts. This collaboration occurred without any money changing hands and resulted in an open source product that met VA's requirements. The finished code was then adopted by VA. The software developer then successfully competed for a support and installation service contract. This OSEHRA-facilitated open collaborative process saved VA at least 18 months of time and at least $2 million. In the

end VA obtained an open source product that fits VA's needs, and the developers created a better commercial product for other customers.

3.4 Software and Tools

3.4.1 Certification

The availability of open source code is increasing rapidly, and it is highly likely that code exists in the open source community that could be incorporated into the efforts described in this RFI. However, there have not been any standards against which the usability of open source code can be assessed. Open source software available online is often difficult to use and is not always reliable. To address such issues in the greater open source community, OSEHRA has developed a certification process that ensures certified software is:

- Usable: Appropriate licensing, documentation, code conventions, and integral tests are included.
- Safe: Individual code units do not cause errors in other components of the system, and the code is robust to all code paths and conditions.
- Compliant: Code meets agreed upon interface specifications and is adherent to all applicable laws and regulations.
- Functional: Code has a defined set of requirements that are met when the code executes.

	Name / Number Space	Dependency / SAC	Open Source License	Documentation	Regression	Code Review	Functional Testing	Test Installation
Level 1	Pass	Pass	OSI-Approved	None	Existing Tests Pass	Large # Non-critical Issues	Large # Non-critical Issues	Large # Non-critical Issues
Level 2	Pass	Pass	Core is Apache 2	Basic	Existing + Some R. Tests	Small # Non-critical Issues	Small # Non-critical Issues	Small # Non-critical Issues
Level 3	Pass	Pass	Core+ is Apache 2	Substantial	Existing + >= 50% Coverage	No Issues	No Issues	No Issues
Level 4	Pass	Pass	All Apache 2	All Required	Existing + >= 90% Coverage	No Issues	No Issues	No Issues

The next step in promoting this common, open source quality baseline will be to use it as the starting point for a national consensus standard. To that end, in 2015, OSEHRA became a Standards Development Organization (SDO), accredited by the American National Standards Institute (ANSI). This accreditation allows OSEHRA to utilize its ANSI-approved operating procedures to document consensus and create open source certification standards, which will raise trust and confidence in newly published code, adding value and strength to the open source community.

3.4.2 Open Source Licensing Support

One of the most important tasks in this RFI is task iv(b), which will have a major impact on VA's ability to share code with the community without extensive redaction. Isolation of VA-specific and sensitive information will reduce the redaction workload and thus the turnaround time for code distribution. In distributing the code, however, VA must begin to recognize the difference between public domain code (written by VA employees in the course of their job-related duties) and open source code (written by VA contractors or outside entities, and licensed under an open source license such as the Apache License Version 2.0). Some previous VA efforts have been deficient in handling the licensing and distribution of code. As a result, the current VA FOIA distribution contains some contractor-developed code that should have been licensed as open source rather than placed in the public domain without attribution.

The refactoring envisioned in this RFI is an excellent opportunity to ensure that the VistA code base is appropriately licensed and attributed. To that end, OSEHRA has produced a licensing guide for contractors in cooperation with the VA TAC. While not authoritative, this guide provides practical advice on ensuring proper licensing of code. We urge VA to make proper licensing and attribution a mandatory requirement in any RFP that includes these RFI tasks and, further, to mandate that all code deliverables (with the exception of configuration files containing VA-specific/sensitive information) be delivered as properly licensed open source code under the Apache License, Version 2.0.

3.4.3 Visualizing VistA and Namespace (ViViaN™)

VistA has evolved to incorporate decades of contributions from practitioners and developers. Its sheer size and complexity, together with the monolithic, namespace-driven nature of its native M Code, makes it extremely difficult to visualize. Several attempts have been made over the years, including a widely-used functional diagram nicknamed the "onion diagram," as well as a limited internal graphical tool.

OSEHRA has taken a leading role in educating the community about VistA, and that mission has led to the requirement for a tool that can:

- Provide an interactive venue to explore the structure of VistA from a developer perspective, using multiple hierarchical approaches such as functional package structure, the roll and scroll menu structure, and the Business Function Framework;
- From any hierarchy, "drill down" to a point where package-level dependencies and actual M Code can be viewed;
- At the package level, expose functionality such as M-APIs, RPC Calls, HL7 Calls, and Web Services;
- Show differences among major VistA distributions such as VA's FOIA VistA, DSS, Medsphere, and others; and,
- Identify gaps in VistA distributions that would need to be addressed for a full commercial implementation, such as the absence of a billing module in the VA distribution.

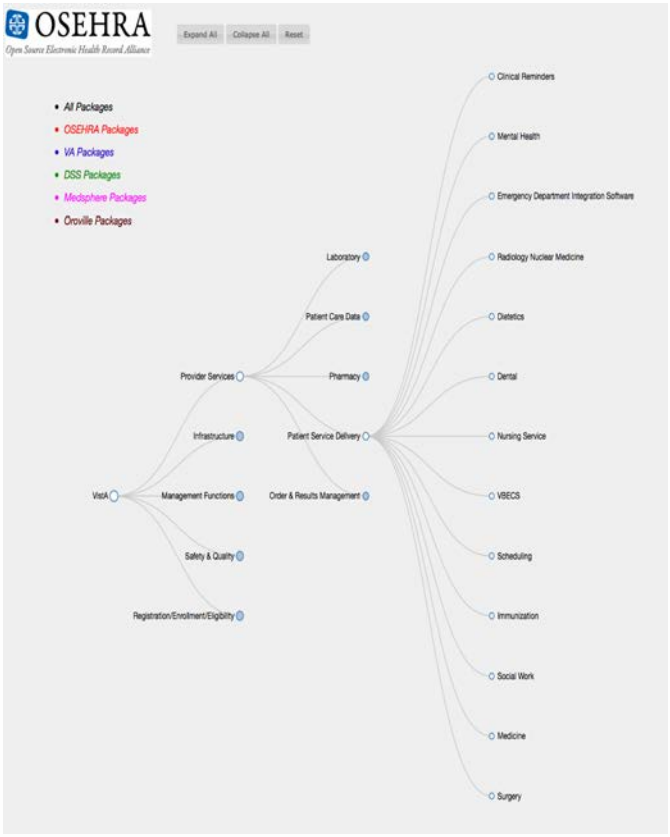

Figure 3-1: A sample ViViAN screen showing expansion of patient service delivery.

Earlier this year, OSEHRA developers identified an open source tool that provided an excellent interactive platform to address these requirements. It currently utilizes a specially-formatted text file to drive an on-line interactive display. See

http://code.osehra.org/Prod/Visual/prod_visual.html for the demonstration. The tool was named ViViAN™, for VIsualizing VIstA and Namespace. An OSEHRA Open Source Project Group (the Visualization Project Group) was established to manage ViViAN using the Agile Methodology. ViViAN was transitioned into production in May 2014, and is currently being maintained by the OSEHRA Visualization Work Group.

3.4.4 Enterprise Level Software Testing Environment

OSEHRA has developed and implemented an enterprise level repository and automated testing infrastructure (see diagram below) to enable the application of the OSEHRA certification standard in maintaining the VistA EHR codebase.

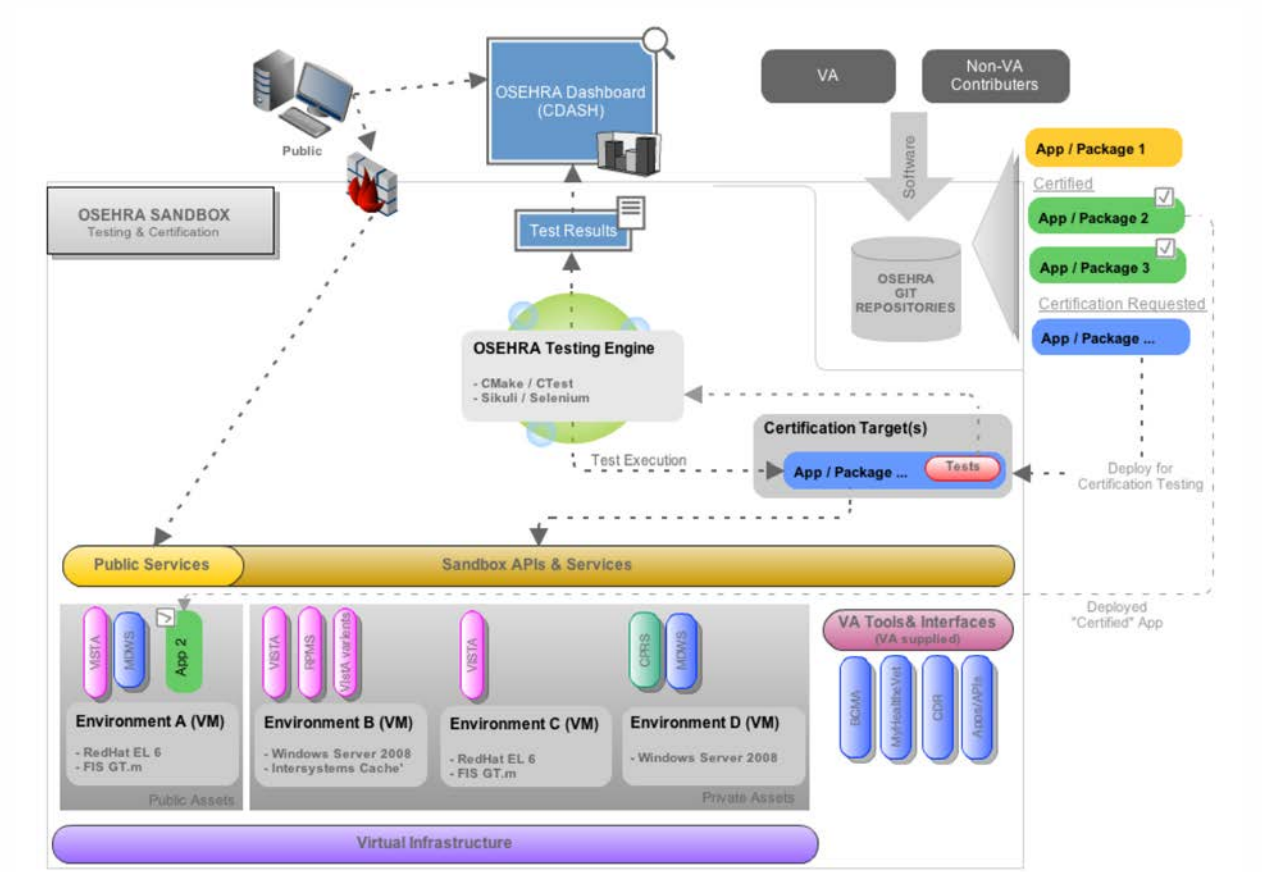

Figure 3-2: OSEHRA Testing Environment

This platform is not intended to replace any portion of VA's internal testing and certification process. Instead, it may provide a useful common test environment for open development and collaboration outside the VA network. It could be utilized to allow contractor(s) and community members to jointly develop and test major portions of the VistA re-factoring effort prior to submission to VA.

Key features of this platform include:

- Automated tool suite for extracting module-to-module dependencies for safe code analysis
- Interoperability/conformance testing against defined system interfaces
- Nightly unit and regression testing/certification against multiple production environments
- Dashboard report on the status of the codebase to include results of unit and regression tests, compliance and functional testing.

For a highly complex system such as VistA EHR, containing 170+ software modules with a large number of module-to-module dependencies, it is difficult to assure that changes to the software will not cause errors in other components. Extensive manual testing would be too

expensive. To ensure new code is safe for OSEHRA certification, OSEHRA developed an automated module-to-module dependency extraction tool to enable quick review of potential code safety issues.

To maintain reusable, easily maintained code, OSEHRA uses automated static code analysis tools to make sure that the code is compliant with community coding standards and conventions. For support of agile development with a short release cycle, regression testing consisting of unit tests and system integration tests with high code coverage is performed nightly against commonly-used production environments. The results are displayed on a dashboard that provide a community-facing report on the status of the codebase that include results of automatic unit and regression tests, as well as compliance and functional testing.

Finally, to allow for greater community participation in software development and testing, OSEHRA created an automated installation and testing of VistA EHR using tools such as Vagrant. This enables potential collaborators to quickly build the development environment in minutes without spending hours to configure the highly complex software.

4 Summary

There is no shortage of challenges in updating and securing VistA. In fact, one of the challenges is a shortage – the expertise required to safely and effectively make large-scale modifications to VistA at the core level is scarce, and getting scarcer. The kinds of tasks described in this RFI will require help from across the entire VistA community. No single contractor (or team of contractors) has sufficient expertise and experience to deliver solid, world-class solutions on all of the specified tasks. *However, a well-qualified contractor team, operating transparently with participation from the open source community, could produce an extraordinary next-generation VistA.*

An open approach will require innovation, both from VA and the contractor(s). VA must commit to working in an open, transparent environment, and demand the same from their prime contractor. Contractors are not used to working openly – they are highly competitive by definition. VA will need to mandate a new development strategy via contract requirements in order to enjoy the benefit of open collaboration.

Appendix A – Recommendations for Modernization of VA Standards and Conventions for Code Generation

OSEHRA

Open Source Electronic Health Record Alliance

OSEHRA VistA Gold Disk Project Support

Proposed Updates to VA SAC for VistA National Software

Delivered in Fulfillment of SLIN 1002AG

Contract Number: VA118-13-C-0008

Version 7.0
December 10, 2014

Revision History

Date	Version	Description	Author
06/11/2013	Draft	Initial Content	Edwards
06/13/2013	1	Deliverable	Edwards/Hewitt
09/13/2013	2	Quarterly Update	Edwards/Hewitt
12/13/2013	3	Quarterly Update	Edwards/McClaughry /Hewitt
03/13/2014	4	Quarterly Update	Edwards/McClaughry /Hewitt
06/13/2014	5	Quarterly Update	Edwards/McClaughry /Henderson/Hewitt
09/15/2014	6	Quarterly Update	Edwards/McClaughry /Henderson/Hewitt
12/10/2014	7	Final	Edwards/McClaughry /Henderson/Hewitt

Table of Contents

1. Executive Summary

The VA Standards and Conventions (SAC) document for VistA specifies how M code should be written and the rules developers should follow. The SAC has been around for many years and has evolved over time as new lessons have been learned and as the M standard has been modified to allow for new coding constructs and commands.

At this time, the M standard has been unchanged for some time, but the two major M vendors (InterSystems Caché and Fidelity Information Services (FIS) GT.M) have both advanced beyond the M standard in their environments. The recommendations detailed in this document will allow for code to be developed that will work on both M vendors' systems.

2. Approach

During the course of this contract, the OSEHRA staff has requested community feedback via the OSEHRA mailing lists, the Hardhats mailing list, and the Architecture Working Group. OSEHRA also reached out directly to groups developing VistA packages in an open source environment to be able to better collect recommendations from the front lines.

As comments arrived from various mailing lists, OSEHRA moved them to its IdeaScale website (http://osehra.ideascale.com/) so the various recommendations could be voted on by the community (more votes = higher ranking). The site also acts as a tracker for recommendations to lessen the chance of duplicates. The recommendations below have been transcribed in the order of popularity on IdeaScale. Most recently, the concepts and recommendations described herein were discussed during the first OSEHRA Code Alignment Workshop in Oakland, California, which was held from November 19-21, 2014. No changes or additions to these recommendations were identified in Oakland – community members expressed the feeling that they had reached consensus, that the input provided here is complete and reflects community opinions. Therefore, this final version of the deliverable has no technical changes.

OSEHRA has omitted detailed examples from the recommendations, since the intended audience of this document (such as developers and the SAC Committee) is already familiar with the constructs described. If detailed examples are necessary, please contact OSEHRA.

3. SAC Recommendations

The following is a summary of the discussion around certain SAC sections and recommended courses of action regarding the change.

3.1. Routine Name Limits

Routine names and labels are limited to eight (8) characters, not including the formal list for parameter passing, and may not contain lowercase characters.

- **Discussion:**
  - This rule restricts the use of variable/routine names that may be more descriptive than can be expressed in 8 characters. In addition, the expanding use of 4 character namespaces complicates variable/routine naming by leaving fewer characters available. This forces programmers to create cryptic, instead of logical, names.
  - Current rules do not permit mixed case for labels and namespaced variables, which further inhibits the ability to incorporate meaning in the names.
  - With no votes against, expanding the routine name limits is the most popular SAC modification as ranked by the community.

- **Recommendation:**
  - Increase allowed length of variable/routine names to 31 characters. Both M implementations allow 31 characters for routine name and label length.
  - Permit mixed case labels and namespaced variables. This allows for the names to be more descriptive and makes it easier to spot the words contained within the description. There will need to be a SAC recommendation to either use Pascal case or Camel case as the standard.
    - Note: In order to fully use this expanded limit, certain input transforms and field lengths would need to be modified to accept the new expanded length. Examples of this can be found within files related to Kernel Installation and Distribution System (KIDS).

- **Relevant SAC Section:**
  - 2.2.4

Sources:

http://docs.intersystems.com/cache20102/csp/docbook/DocBook.UI.Page.cls?KEY=GMSM_architecture

http://tinco.pair.com/bhaskar/gtm/doc/articles/GTM_V5.4-002A_Supplementary_Information.html - id1611608

http://osehra.ideascale.com/a/dtd/Increase-limit-for-routine-and-label-names/478506-23465

3.2. Restrictions on Using Lower Case

- **Discussion:**
  - Most programming languages today allow the use of lower case in the source code.
  - Permitting use of lowercase characters throughout the M source code would draw the developer's attention to important places or words capitalized within the source code.
  - There is no technical restriction on the M implementation side, as both M implementations permit lowercase within the M source code.
  - There is one reservation about including lowercase characters in routine names: when VMS is used, it automatically uppercases all filenames, which will cause problems with GT.M on VMS, however GT.M on VMS is no longer receiving enhancements by the developer.

- **Recommendation:**
  - Remove the restriction on using lower case.

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Allow-for-lowercase-throughout-M-code/478551-23465

3.3. Allow mixed case Labels, Variables, and Global Subscripts

- **Discussion:**
  - Allows for easier code reading (names stick out in variables)

- **Recommendation:**
  - Allow mixed case labels, variables, and global subscripts.

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Allow-mixed-case-labels-variables-and-global-subscripts/478560-23465

3.4. Standardizing APIs for Various Formatting Cases

- Discussion:
- The kernel provides many utility functions that various VistA packages choose to implement themselves. For example: there are several uppercase APIs throughout the code that perform a $TRANSLATE.
- Since there is a kernel API that performs this function, the various VistA packages shouldn't create their own implementation of the uppercase API.

- **Recommendation:**
  - The SAC should enforce that packages/developers use a standardized API for this activity. This would allow for a single API change to allow UTF-8 character support that may be required to represent patient names correctly.

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Standardize-APIs-for-Uppercasing-Date-Formatting-etc/614707-23465

3.5. Maximum Routine Size Limit

The maximum routine size, as determined by executing ^%ZOSF("SIZE"), is 20,000 characters. 15,000 of the allowed 20,000 characters may be non-comment source lines (including ";;" comment lines). 5,000 characters are reserved for non-";;" comments.

- **Discussion:**
  - This restriction results in artificially limiting comments within routines, whether or not the intention exists. A better rule might be to limit routines to cover only one logical separation of the code. This would enable each routine to contain all of the processing needed for one logical separation in one place, rather than forcing it to be continued in another routine or otherwise split apart in illogical ways.

- **Recommendation:**
  - Revise the limits on routine size to increase logical coherence and to encourage thorough comments for code documentation.
  - The maximum length of a routine should be the logical end of the routine.

- **Relevant SAC Section:**
  - 2.2.7

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Remove-routine-length-limits/482554-23465

3.6. Standardize Error Handling

- **Discussion:**
  - $ZTRAP is not part of the ANSI standard for M, and has been replaced with $ETRAP.
  - VistA currently uses both $ZTRAP and $ETRAP. This can also cause unpredictable results along with inconsistencies between M implementations.
- Recommendation:
  - Any new code, as well as any code undergoing significant revision, should use $ETRAP to handle errors and $ECODE to throw them (see 3.1.8 below).

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Standardize-on-$ETRAP-$ECODE-error-handling/605876-23465

3.7. Routine First Line Restriction

The first line of a routine cannot contain the formal list for parameter passing.

- **Discussion:**
  - Some open source community members would like this restriction removed to allow for easier and faster calling of the routine with parameters, removing the need to use a label for the only externally callable entry point.

- **Recommendation:**
  - The restriction can be relaxed for single purpose routines only, while leaving the restriction in place for any multi-purpose routines.
  - Some community members hold to requiring the first 2-3 lines only contain metadata about the routine and no calling information.

- **Relevant SAC Section:**
  - 2.2.1.1

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Allow-parameters-to-be-passed-on-the-first-line-of-a-routine/478515-23465

3.8. Standard Error Codes Produced by Application Code

- **Discussion:**
  - There should be a place where application code that standardizes throwing an error using $ECODE with values such as ",U[number]," can register what [number] means.

- **Recommendation:**
  - Work with the DBAC to create a registry of standardized error codes, so that U1, for example, would have a standardized meaning across all packages, while allowing for some space for package specific error codes.

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Standard-set-of-error-codes-generated-by-application-code/605880-23465

3.9. Use of Long Global Nodes

  - Discussion:

- The current SAC requires adherence to the 1995 M standard, which limits global nodes to 255 characters. However, recent medical coding standards have longer descriptions and more space in the global nodes would better accommodate these standards. Other popular M applications like RPMS (Resource and Patient Management System) and EWD.js have taken advantage of longer global nodes to support their applications.

- **Recommendation:**
  - Increase limit for global nodes to 32,000 characters, as this is the maximum length granted by GT.M and InterSystems Caché

- **Relevant SAC Section:**
  - 2.1

- **Sources:**
  http://71.174.62.16/Demo/AnnoStd?Frame=Main&Page=a202015&Edition=1995
  http://osehra.ideascale.com/a/dtd/Allow-long-global-nodes/480667-23465

3.10. Remove Restrictions on End of Line Whitespace

- **Discussion:**
  - When modifying or creating code in an editor, the restrictions on spacing can cause the routine to fail the XINDEX code checker, and force programmers to go through line by line to find the end of line whitespace.
  - The current M standard does not require removal of all trailing whitespaces, to the best of OSEHRA's knowledge.
  - The community has valid points about removing the restriction, but OSEHRA's observation is that other source code (outside of M) enforces the same trailing whitespace restriction, including languages such as Java, Python, etc. OSEHRA's own repositories are set up by default to not allow trailing whitespace and allow it only on exempted files where it may be necessary.

- **Recommendation:**
  - Retain current SAC requirement that end of line whitespace be removed.

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Remove-restriction-on-terminal-spaces-on-line/624862-23465

3.11. Assumed Variables

- **Discussion:**
  - Assumed variables have no formal declaration in parameter lists for a label or routine.

- The use of assumed variables makes it dangerous to work or perform maintenance on the software.
- A significant portion of the open source community would like to see fewer or no assumed variables.

- **Recommendation:**
  - Restrict or forbid the use of assumed variables in newly created or refactored code.
  - A possible restriction is to limit assumed variables to DT, DUZ, and U. This was the practice that the VistA Code Refactoring Team used in their refactored code.

- **Relevant SAC Section:**
  - 2.3.1

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Limit-assumed-variables/482378-23465

4. Programming Best Practices Recommendations

4.1. Modularized Code

- **Discussion:**
  - Modern programming standards have started to migrate to being API-centric rather than UI (User Interface) focused. By creating M functions that have well-defined parameter lists, proper scoping of variables, and do not require any user interaction, VistA applications can become modular in nature and allow for reuse in other applications.

- **Recommendation:**
  - Provide developer training and outline in the SAC that functionality should be written in a modular fashion.

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Think-in-functions-not-procedures/482526-23465

4.2. Routine Argument style

- **Discussion:**
  - Using arrays as arguments to a function is a modern, object-oriented approach to programming. It is more natural to pass multiple arguments in one array rather than using a large number of literals.

- **Recommendation:**
  - Encourage the use of arrays as arguments, which allows for arguments by name rather than relying upon positional arguments.

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Use-arrays-as-arguments-rather-than-parameters/482529-23465

4.3. Conditional Statements

- **Discussion:**
  - Conditionals should be expressed using the "IF DO, ELSE DO" with dot blocks syntactical convention in order to prevent unpredictable effects caused by $TRUTH.

- **Recommendation:**
  - Although the issue is listed here based upon community responses, OSEHRA recommends that this issue be handled through developer training and through standardization during the code review process. Code review should include a step to ensure developers do not write needlessly complex code that depends upon the $TRUTH variable.

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Standardize-conditional-statements-with-dot-blocks/482546-23465

4.4. Naked References

- **Discussion:**
  - Naked references are basically developer shorthand, but they increase the complexity of code while decreasing readability and maintainability. The SAC already tries to limit the usage for naked references, but some members of open source community believe naked references should be removed entirely.

- **Recommendation:**
  - Recommend against using naked references except where required (like in FileMan screens, etc.). The SAC is already fairly restrictive with regards to naked references in routines

- **Relevant SAC Section:**
  - 2.2.10

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Forbid-the-use-of-naked-references/482549-23465

4.5. Use of $TEXT

- **Discussion:**
  - $TEXT is commonly used to store data within routines or to check for the presence of another routine. There have been many creative ways of using $TEXT that should be avoided, as such uses increase maintenance costs or circumvent the use of standard VistA structures for storing or transporting data.
  - For example, $TEXT is frequently used to transport the data to a new system. This is generally safe; however, data used in a VistA instance should be moved to a FileMan file so that it can be updated independently of the routine that holds the data. Using $TEXT in this fashion complicates maintenance and should be avoided.

- **Recommendation:**
  - The community has reached consensus at the time of writing this document.
  - OSEHRA will withhold comments on recommendations until more feedback is available.

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Restrict-usage-of-$Text/482376-23465

4.6. Use of $ORDER

- **Discussion:**
  - Standardization would make the $ORDER construct easier to understand in the future, rather than allowing every developer to establish his or her own method.

- **Recommendation:**
  - The open source community raised this issue, but the members are not unanimous on the need for action. OSEHRA has no official stance on this issue.

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Standardize-$Ordering-through-a-global/482544-23465

4.7. Use of Indirection

- **Discussion:**
  - Indirection is an incredibly powerful tool that, when used incorrectly, can increase the difficulty of ongoing development efforts.
    An example that should be prohibited is N @A, which does not make the intended purpose of the code immediately obvious.

- **Recommendation:**
  - Limit use of indirection to infrastructure code and, when used, provide comments to clearly document the indirection use.

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Limit-usage-of-indirection-@/482380-23465

4.8. Allow Only One Command per Line

- **Discussion:**
  - The M language is very flexible in the style of programming used and accepts both the in-line or block style of programming. The in-line style of programming has advantages when memory and disk space are at a premium; however the modern computers easily execute the more verbose and user-friendly block style.

- **Recommendation:**
  - OSEHRA does not recommend changes to current practices. Most of the community has voted against this idea and currently has negative votes.

- **Sources:**
  http://osehra.ideascale.com/a/dtd/One-command-per-line/482533-23465

4.9. Use of Post Conditionals

- **Discussion:**
  - Post conditionals make code harder to read due to the implied IF statement tacked on to the ends of commands.

- **Recommendation:**
  - OSEHRA does not recommend changes to current practices. Most of the community has voted against this idea and currently has negative votes.

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Forbid-the-use-of-post-conditionals/482540-23465

4.10. Use of $SELECT

- **Discussion:**
  - The $SELECT function is a powerful tool. However, it shares pitfalls similar to the post conditionals mentioned above, including unpacking all of the implied IF statements and making debugging a more time consuming process. The logic would be far clearer using IF statements rather than the $SELECT function.

- **Recommendation:**
  - OSEHRA does not recommend changes to current practices. Most of the community has voted against this idea and currently has negative votes.

- **Sources:**
  http://osehra.ideascale.com/a/dtd/Forbid-use-of-$Select-command/482534-23465

5. Redaction

5.1. Redacted Hard-Coded Items

- **Discussion:**
  - The OSEHRA community deals only with redacted code, and the VA's process of redaction has historically worked more against the community than with it. Typically, these issues revolve around specific sections where redaction has removed hard-coded assumptions within the code that are required for proper functionality.

- **Recommendation:**
  - Encourage developers to use the tools available in VistA to transition historically hard-coded items to configuration data. These parameters can include:
    - IP/DNS addresses
    - Ports
    - Usernames
    - Passwords
    - VA locations
    - Mail groups
  - Parameterizing the above data and removing the hard-coded dependencies can simplify the software. Additionally, it allows the OSEHRA community to use reasonable substitutes for VA redacted values, while allowing the VA and other implementers to maintain the configuration parameters required in their respective VistA environments.

5.2. Redacted APIs

- **Discussion:**
  - There are instances where parameters will not add the loose binding that is required, such as the implementation of non-redistributable algorithms or applications. (For example, the XUSRB1 routine which is redacted in the FOIA release.)

- **Recommendation:**
  - Implement the non-redistributable algorithms or applications as APIs, so the non-redistributable code can be sufficiently encapsulated and allow for alternative implementations to be inserted when needed. This would ease VA's redaction effort and allow for easier code intake, as only the "pluggable" portion of the application would need to be modified and not the application core.
  - For items that cannot be pluggable, implement a redaction tag strategy. Code to be redacted would be surrounded by "; REDACT" for M code and something similar for other programming languages, so an automated tool could search for those tags and remove the protected code. It is vital to include a description of the redaction so the

community understands what the redacted code does and how to replace it with an alternate implementation. It is important minimize redaction tags use, and rely on parameters or pluggable APIs, as described above, wherever possible.

- Redaction, from OSEHRA's point of view, should remove the implementation but leave the input and output descriptions and variable so an API is effectively created. This approach will prevent open source VistA from becoming an unmaintainable fork, causing VA to lose access to open source innovations.

5.3. External Dependencies

- **Discussion:**
  - VA does not consistently provide documentation about external dependencies (ex: spring, log4j, etc.) in use within a project. This frequently results in compiling source code becoming a task of looking at error messages of missing dependencies or the user being unable to figure out the required version. In the open source community, it is common to have a file, such as Apache 2.0's NOTICE file, that details any included projects and the licenses for those redistributed items, allowing external developers to know what dependencies are in use and whether the source code can be redistributed. Including such a file would benefit VA when performing redaction for FOIA releases, making it clear what can and cannot be redistributed.

- **Recommendation:**
  - Include references to externally retrieved dependencies, including versions and licensing requirements.

5.4. Removal of Non-redistributable/Proprietary Data

- **Discussion:**
  - Wholesale removal of non-redistributable data, including the schema, is the current VA redaction practice.
  - OSEHRA and the open source community understand that not all data and programs are going to be Free/Open Source 100% of the time, and some kind of license fee may be required to use some software or data. Knowing where to access the software/data and the schema allows the community to recreate the functionality that was redacted.

- **Recommendation:**
  - Retain the schema when removing non-redistributable data (ex: CPT codes) so that the application can be used as intended after any required licenses are acquired.

6. Testing

6.1. Unit Testing

The VA performs both manual and “trust” forms of testing, however VA does not currently perform unit testing on code produced in the M language.

- **Discussion:**
  - Unit testing is very effective in ensuring that bugs that have occurred in the past do not reoccur (regression testing) in future versions of the program. The checks can be done whenever the unit test suite is run against the code in question and gives a simple yes/no answer.
  - Unit testing can also provide examples of how to use APIs that are exposed by the application so subsequent developers know how to use the API to get the data they need, while the original developers know that the APIs return the intended data.
  - The VA currently performs this variation of testing on various programming languages it uses in-house, however does not do so for M code

- **Recommendation:**
  - Add unit testing to new releases of M code to prevent broken code from being released, helping to ensure that future releases are more stable and error free.

7. Code Documentation

Other programming languages allow for documentation to be written within the code and extracted out into other formats, such as HTML and PDF. This methodology would be useful for documenting APIs, RPCs and other programming constructs. VistA has many places this documentation could go and OSEHRA would like to reserve the privilege of making recommendations on placement and appearance to ensure ease of location and retrieval for future users, as well as complying with best practices in software development.

Various code style guides include definitions and requirements for in-line code documentation to document methods, parameters, and outputs. Popular in-line code documentation tools include Doxygen, JSDoc, pydoc, and javadoc. By using in-line code documentation and these tools, automated code documentation can be produced. This will help developers understand the code better by listing the input, output and a general description of the method. MUMPS, like several other languages, doesn't have any built-in support for these tools, but it is fairly easy to add this capability to existing source code as the real change is having structured comments.

For VistA the following style is an initial proposal that will be discussed and expanded upon with the OSEHRA community:

- Before every label have the following structured comment:

  ;; This method creates an allergy for a patient. This is an extrinsic function
  ;; @param RETURN – Variable that will contain the results of the RPC call in the
  ;; format:
  ;; RETURN(0)=success (0=false,1=true)^user friendly text(Allergy BEE STING
  ;; Added)
  ;; @param DFN – Patient IEN
  ;; @param ALLERGEN – Pointer to File/IEN of allergen
  ;; @param ORIGIN – Date/Time of when the allergy is documented (defaults to visit
  ;; date
  ;; @param HISTORICAL – Flag 0=False, 1=True if this is a historical documentation
  ;; @rpc ORWDAL32 SAVE

This documentation format provides an easy-to-read explanation of what the code is for, parameter definitions and functions, variable formatting and what RPC is represented. By documenting every parameter, its format, and type the intended use of this code is very obvious even to non-programmers.

When converted to external format (HTML/PDF) the documentation becomes an API guide that will help new developers explore the capabilities of the code before they even read the actual code, or for interface developers who only need to know the information contained within the structured comments.

8. Future Enhancements

OSEHRA would like to make further recommendations in the documentation of routines and creation of rules for development in the future:

- XINDEX needs to be modified whenever the SAC is modified. This will minimize the number of 'false' error indications returned by XINDEX and allow for improved certification of VistA code. OSEHRA believes that this was VA practice in the past, but is not currently part of the official change procedure. For example, the current version of the SAC allows lowercase variables if they have been "newed" in that routine, but XINDEX will still mark each occurrence as a SAC violation.
- Continue to reduce the number of hard-coded assumptions throughout the code (such as calls to internal VA services, locations, and names) and moving more of these functions to configuration parameters.
- Expand the dialogue for establishing standards and conventions to be more bidirectional, with better feedback and discussion about the recommendations, would also contribute greatly to OSEHRA's ability to provide support to the community and to the VA.
- OSEHRA is participating in discussions with VA regarding redaction including: delivering whitepapers, recommendations for how to handle redaction (including this document), and other discussions where redaction is relevant.

9. Continuing Conversation

While this is the final deliverable of this document under the Gold Disk Support contract, OSEHRA plans to continue to monitor its IdeaScale website, workgroups, and HardHats mailing list posts for relevant ideas. In addition, these recommendations will be discussed in OSEHRA Working Group meetings and Code Alignment Workshops. We will continue to provide feedback and recommendations to VA via other channels following the completion of this contract. Online resources include:

- http://osehra.ideascale.com/
- https://groups.google.com/forum/?fromgroups - !topic/Hardhats/XJSx5OuKMLE
- http://www.osehra.org/content/opportunity-contribute-sac