**VistA Application Developer Documentation**

**Introduction**

Goal

A key goal of the VistA Application Developer Documentation is to empower external developers with little to no experience with VistA to build innovative applications on top of VistA using mainstream technologies like JavaScript.

Audience: Software Developers

Document Purpose

This documentation provides instructions on how an external innovator can assemble existing tools to establish a development environment for building VistA applications. This documentation provides instructions for a developer on setting up a basic local development environment along with sample applications. Also included is reference to other resources, like relevant developer tools and forums for developer discussion.

Document Organization

The rest of this document is organized as follows:

1. Setup of Development Environment
2. VistA Application Programming Interface
3. Catalogue of Developer Tools
4. Forums, Lists and Blogs
5. Frequently Asked Questions
6. Invitation for Contributions

**1. Setup of Development Environment**

This document describes three development environments that a developer can use to create VistA applications:

1. VistA Novo Test Stub (local)
2. Open Source VistA Environment (local)
3. VA VistA Environment (centralized cloud)

The environment that a developer chooses to use depends on whether the intent is to deploy the VistA application to an open source VistA or VA VistA production environment. A developer can also start to develop a VistA application in a local environment and over time progress to using a cloud environment for further development and testing, as illustrated in Figure 1 below.



Figure 1. Relationship between Local and Centralized VistA Application Development Environments

**Considerations in Choosing a Development Environment**

The VistA Novo Test Stub is the easiest environment to get up and running quickly. It provides the greatest flexibility with which to develop a VistA application. However, in order for the VistA application developed using the Test Stub to operate in a production environment, VistA-based services mocked by the Test Stub must exist in the production environment.

The Open Source VistA Environment provides the benefit of having a local VistA environment that is fully functional and under the control of the developer. For example, a developer is able to load test data into the system for development and testing of the VistA application. In order for the VistA application to operate in a production environment, however, there must be a means for the application to interact with the production VistA system through a well-defined VistA interface.

The VA VistA environment makes available the VA VistA environment to developers to implement VistA applications. It enables a developer that does not have the ability to stand up a comparable local environment (for example, due to software licensing issues) to test their VistA application within a VA VistA environment. VistA applications targeted for use by the VA should be at the very least tested within this environment.

This section provides instructions on setup and use of each development environment.

**VistA Novo Test Stub**

Overview

VistA Novo is a proof of concept open source VistA developer toolkit that allows developers working in the open source community to use and contribute VistA capabilities using mainstream architectural approaches and modern programming languages.

VistA Novo provides a test stub that mocks existing or prospective VistA-based services intended to simplify testing, particularly hard to generate corner cases and error conditions. The test stub enables parallel development of applications while VistA-based services are being developed. Applications can access data using Fast Healthcare Interoperability Resources (FHIR), an emerging HL7 open standards-based format.

For more information on VistA Novo, visit the [VistA Novo Project Website](http://osehra.org/content/vista-novo-project-overview).

Basic Setup

Instructions on installing and setting up the VistA Novo test stub are available [here](https://github.com/OSEHRA/vista-novo-test-stub/blob/master/README.md#installation-and-setup).

Test Data

Instructions on creating and populating a data model in the VistA Novo test stub are available [here](https://github.com/OSEHRA/vista-novo-test-stub/blob/master/README.md#service-creation).

Sample Application

Accompanying the VistA Novo test stub is a blood pressure demo application that demonstrates the functionality of the test stub through a simple blood pressure use case. The demo application is available [here](https://github.com/OSEHRA/vista-novo-demos).

**Open Source VistA Environment**

Overview

There are four basic components to the open source VistA environment: OSEHRA VistA, Computerized Patient Record System (CPRS), Enterprise Web Development and Test Data, as illustrated in Figure 2.

 .

Figure 2. Components of the Local VistA Application Development Environment

OSHERA VistA is an open source Electronic Health Record (EHR) system that is based upon the VistA EHR system developed by the VA. The system comprises a core platform and a collection of interoperable components that allow implementers to assemble a combination of features and functions required. OSEHRA VistA runs on top of the [GT.M](http://en.wikipedia.org/wiki/GT.M) database engine. More information on OSEHRA VistA is available [here](http://www.osehra.org/content/osehra-vista).

Enterprise Web Development (EWD) is an open source product that operates on top of OSEHRA VistA (and theoretically with VA VistA) to develop web-based applications, combining a back-end MUMPS data management environment with browser-based client-side graphical user interface. EWD adds basic connectivity to VistA through the Node.js JavaScript platform. It includes a demo website for displaying simple patient data. For more information on EWD, visit the [EWD Reference Guide](http://gradvs1.mgateway.com/download/EWDjs.pdf), [The EWD Files](http://robtweed.wordpress.com/), and the [EWD Code Repository](https://github.com/robtweed/ewd.js).

Basic Setup

Setting up an Open Source VistA environment entails installing [OSEHRA VistA](http://www.osehra.org/content/osehra-vista) and EWD. OSEHRA VistA can be installed as a virtual machine. Instructions for doing so can be found [here](https://github.com/OSEHRA/VistA/blob/master/Documentation/Install/Vagrant.rst). Alternatively, a developer can review the [5-part series](http://osehra.org/blog/ewdjs-and-vista-ready-use-30-minutes) on how to get EWD.js and VistA up and running.

The installation should take roughly 30 minutes.

Note:

* The installation process talks to several external servers and might not work correctly behind a proxy, so if the installation is unsuccessful, try connecting directly to the Internet.
* On some machines, the default Video Memory (VM) setting of 12 MB is insufficient, and the installation fails. If this happens, you can stop the virtual machine from within VirtualBox, change the settings for the VM, and then continue the installation process via **vagrant up**.

To verify that the installation was successful, visit <https://localhost:8080/ewd/VistADemo/index.html> in your browser. If the installation was successful, you should be prompted for an access code and a verification code. You can use the following credentials: **fakedoc1**/**1Doc!@#$**. These credentials are from the OSEHRA VistA installation guide.

Test Data

There are no known synthetic patient datasets available that can be loaded automatically into OSEHRA VistA. The only option to load data into OSEHRA VistA is through the Computerized Patient Record System (CPRS). CPRS is a standalone graphical user interface for VistA that allows one to manually add/remove/update sample patient data. On a Windows machine, you can install CPRS using the installation instructions at <http://osehra.org/document/guis-used-automatic-functional-testing>. You'll need to change the specified port to 9430, e.g. **"Path/to/CPRS" P=9430 S=127.0.0.1**.

Sample Application

There is a VistA Demo application that comes with EWD. A technical deep-dive into this demo application is available [here](http://robtweed.wordpress.com/2014/03/02/ewd-js-vista-a-technical-deep-dive-into-the-vista-demo-application/).

RCPClinicalHeadings is another sample application that demonstrates how to interact with VistA using JavaScript functions that map to one or more M routines on the back-end. The open source software for this sample application is available [here](https://github.com/borochris/RCPClinicalHeadings/tree/d63451d834ed4b0caa53d4c3ba757d13de721b79).

The following EWD functions have been exposed via a web interface in this sample application:

* [getDemographics](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L487-L499) : Basic patient demographic information
* [listComplaints](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L261-L273)/[getComplaint](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L274-L283) : Presenting complaints/issues
* [listProblems](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L337-L349)/[getProblem](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L350-L359) : Problems, issues, and diagnoses
* [listProcedures](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L360-L372)/[getProcedure](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L373-L382) : Procedures
* [listOrders](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L450-L476)/[getOrder](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L477-L486) : Lab orders
* [listVitals](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L307-L326)/[getVital](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L327-L336) Vital signs
* [listMedications](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L383-L395)/[getMedication](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L396-L405) : Medications
* [listAllergies](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L428-L439)/[getAllergy](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L440-L449) : Allergies
* [addAllergy](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L500-L515) : Allergies (write)
* [listAlerts](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L406-L417)/[getAlert](https://github.com/borochris/RCPClinicalHeadings/blob/d63451d834ed4b0caa53d4c3ba757d13de721b79/nodeVista.js#L418-L427) : Safety alerts

These methods invoke [custom MUMPS code](https://github.com/borochris/RCPClinicalHeadings/blob/master/ZZCPCR00.m) written to work alongside the above functions. A common pattern includes a "get" method for retrieving a single item by ID, and a "list" method that retrieves all associated IDs for a type of item and then invokes the corresponding "get" method to return all items. There also exists an "addAllergy" method which is the only method included that can write back to VistA.

The installation instructions can be found within the [README](https://github.com/borochris/RCPClinicalHeadings/blob/master/README.md). Additional background information regarding Creating/installing a EWD.js applications can be found in the [EWD.js documentation](http://gradvs1.mgateway.com/download/EWDjs.pdf) under the title "Creating an EWD.js Application". When installing within the OSEHRA VistA environment outlined above, the files in RCPClinicalHeadings should be installed within the VM as follows:

index.html -->> /home/osehra/ewdjs/www/ewd/VistaRCP

app.js -->> /home/osehra/ewdjs/www/ewd/VistaRCP

nodeVista.js -->> /home/osehra/ewdjs/node\_modules

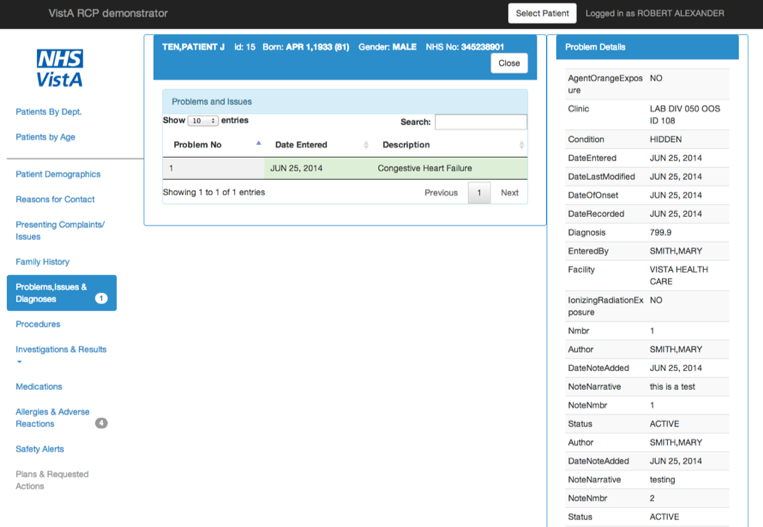
VistaRCP.js -->> /home/osehra/ewdjs/node\_modules

ZZCPCR00.m -->> /home/osehra/p

NHS\_VISTA.jpg -->> /home/osehra/ewdjs/www/images (you may have to create this directory)

One method for transferring files to the VM is to use Vagrant's support for [synced folders](https://docs.vagrantup.com/v2/synced-folders/).

Once you have installed the app, you should be able to reach it at <https://localhost:8080/ewd/VistaRCP/index.html>. You can log on with the same VistA credentials you use when connecting to CPRS or other EWDjs VistA apps. After logging in, you can navigate on the left-hand side through the functionality mentioned above, and you should see something like the screenshots in Figure 3:



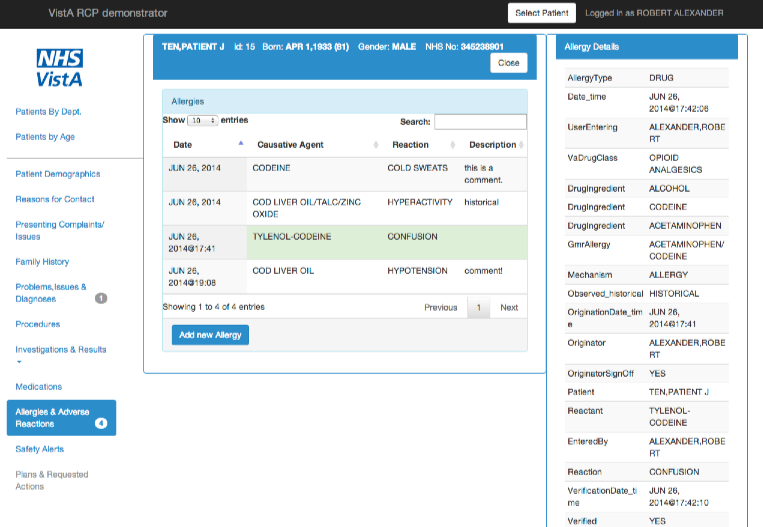


Figure 3. VistA RPC Demonstrator Screenshots

Any information visible in CPRS is also visible via the web interface of this application.

**VA VistA Environment**

Overview

The VA Sandbox Cloud (VASC) provides access to the VA VistA environment on a virtualized infrastructure for development purposes. Software in VASC includes VA VistA and the underlying technology stack on which VistA operates at the VA, including InterSystems Caché which is a proprietary solution with associated licensing cost. There is a public VistA server with a Medical Domain Web Services (MDWS) and Remote Procedure Call (RPC) interface for applications to interact and test against the deployed instance of VistA. In addition to the existing software, developers can also use tools they choose, but anything that they use is subject to VA approval to be allowed back into the VA. Also included are supporting tools for collaboration, idea management, blogging, and source code management.

For more information on VA Sandbox Cloud, visit <https://vacloud.us>.

Test Data

VASC includes a moderate size dataset of 200+ synthetic patient records that are available for development and testing purposes. Note that this data is only available within the VA Sandbox Cloud and only for Caché, i.e. the data cannot be imported in its current form into GT.M and thus the data cannot be used to populate a local open source VistA instance.

Sample Application

There are currently no known sample applications that run within the VA VistA environment.

**2. VistA Application Programming Interface**

A well-defined VistA Application Programming Interface (API) implemented consistently across the three VistA environments is critical to ensure VistA applications are compatible and that they use authoritative VistA business logic so they can securely work within a production environment. The API should provide a layer of abstraction on top of VistA so that developers can implement VistA applications without needing to understand the details of VistA M routines.

Presently, the only API available is via MUMPS/RPC Broker. This is not appropriate for use by a developer that does not have VistA experience. While the RPC Broker is available in several languages (including Delphi, C#, Python), VistA knowledge is needed to understand the data coming back. For such a developer, the VistA API should be implemented as a set of RESTful services with the data structured in the Fast Healthcare Interoperability Resource (FHIR) format - an emerging HL7 standard - such that the VistA application developer interacts with healthcare domain objects rather than needing to work with a sequence of M routines. The VistA application would call the underlying M routines via the abstraction layer. The added benefit of a FHIR-based API is that it’s understandable by both technologists and domain experts.

While currently no such VistA API exists, there are a number of active projects to implement a FHIR interface for VistA that should work together to ensure consistency in implementation. These projects include:

* The [VistA Service Assembler](http://www.osehra.org/blog/minutes-awg-meeting-4-29-2014-vista-service-assembler-vsa-update) (VSA) is potentially an enabling tool to generate VistA based services that use authoritative VistA business logic and provide a layer of abstraction on top of VistA.
* The Enterprise Health Management Platform (eHMP) includes an effort to implement a FHIR interface on top of FileMan and the translation logic required convert the data from VistA into the appropriate FHIR model formats.
* The [EWD.js REST interface](http://robtweed.wordpress.com/2014/03/02/ewd-js-and-vista-rest-interface/) provides a REST interface to EWD.js-based methods, converting REST requests into corresponding digitally-signed EWD.js HTTP(s)-based web service requests.

**3. Catalogue of Developer Tools**

There are a number of ongoing projects that support the development of VistA applications adopting mainstream architectures, programming languages and technologies. These tools provide enabling capabilities that are good for an application developer to understand, though not all developers may use all of these tools for application development.

**GT.M**

|  |  |
| --- | --- |
|  |  |
| Website URL | [www.fisglobal.com/products-technologyplatforms-gtm](http://www.fisglobal.com/products-technologyplatforms-gtm) |
| License | Affero General Public License (AGPL) v3 |
| Maintaining Organization | Fidelity Information Services, Inc. (FIS) |
| Last Activity | Recent |
| GT.M Compatibility | N/A |
| Caché Compatibility | N/A |
| Programming Language | MUMPS, C++ |
| VistA Awareness | No |

GT.M, short for Greystone Technology/M, is an open source MUMPS interpreter and a functional replacement for Caché, a commercial MUMPS interpreter from InterSystems, Inc. GT.M is widely used by open source VistA, including WorldVistA, OpenVistA, vxVistA, and OSEHRA VistA, as well as by the financial industry.

**EWD**

|  |  |
| --- | --- |
|  |  |
| Website URL | [www.mgateway.com](http://www.mgateway.com) |
| License | Affero General Public License (AGPL) v3 |
| Maintaining Organization | M/Gateway Developments, Ltd |
| Last Activity | Recent |
| GT.M Compatibility | Yes |
| Caché Compatibility | Yes |
| Programming Language | MUMPS/Javascript |
| VistA Awareness | No |

Enterprise Web Development, or EWD, is an open source replacement for InterSystems Caché Server Pages (CSP) and allows the use of MUMPS to render HTML. EWD is widely used by open source VistA, including WorldVistA, OpenVistA, and vxVistA.

**MedCafé**

|  |  |
| --- | --- |
|  |  |
| Website URL | [www.medcafe.org](http://www.medcafe.org) |
| License | Apache 2.0 |
| Maintaining Organization | MITRE |
| Last Activity | October, 2011 |
| GT.M Compatibility | Yes |
| Caché Compatibility | Yes |
| Programming Language | Java |
| VistA Awareness | Yes |

MedCafé is a clinical web user interface that can pull data from VistA, using a composable software framework that allows a user to build a personal view of a patient record using a set of tools called components. Each component has simple but separable functionality. By isolating each component’s capability, a component can be added or removed from the system without impacting the functionality of the system as a whole. This means, MedCafé provides the flexibility to adapt to a health care professional’s immediate needs and may, in fact, be used by a wide range of users from clinicians to technicians to the patients themselves.

**DivConq**

|  |  |
| --- | --- |
|  |  |
| Website URL | [www.divconq.com](http://www.divconq.com) |
| License | Apache 2.0 |
| Maintaining Organization | Andy White |
| Last Activity | January, 2012 |
| GT.M Compatibility | Yes |
| Caché Compatibility | Porting effort required |
| Programming Language | Java, MUMPS |
| VistA Awareness | Yes |

DivConq is a MUMPS to Java bridge that allows Java programmers access to MUMPS data. It supports creation, retrieval, updating, and deletion (CRUD) as well as querying on MUMPS data. DivConq works with Java Standard Edition, but can also run optionally with Java Enterprise Edition. Features like auditing, replication, and tracking values over time, and data locking are also supported.

**NodeM**

|  |  |
| --- | --- |
|  |  |
| Website URL | [www.github.com/dlwicksell/nodem](http://www.github.com/dlwicksell/nodem) |
| License | Affero General Public License (AGPL) v3 |
| Maintaining Organization | Fourth Watch Software, LLC |
| Last Activity | Recent |
| GT.M Compatibility | Yes |
| Caché Compatibility | In progress |
| Programming Language | C++, MUMPS |
| VistA Awareness | No |

NodeM is a set of Node.js drivers for MUMPS and integrates Node.js with the underlying database, allowing MUMPS data to be accessed from JavaScript through first-class JavaScript objects. NodeM uses GT.M’s C-based call-in interface to access MUMPS data.

**EsiObjects**

|  |  |
| --- | --- |
|  |  |
| Website URL | [www.esiobjects.org/index.html](http://www.esiobjects.org/index.html) |
| License | Mozilla Public License v1.1 |
| Maintaining Organization | ESI Technology Corp |
| Last Activity | September, 2012 |
| GT.M Compatibility | Yes |
| Caché Compatibility | Yes |
| Programming Language | MUMPS, Java |
| VistA Awareness | Yes |

EsiObjects is an object-oriented application development environment and is a 1995 ANSI Standard MUMPS-compliant, object-relational database management and interoperability system.

**MDWS**

|  |  |
| --- | --- |
|  |  |
| Website URL | [www.osehra.org/group/mdws](http://www.osehra.org/group/mdws) |
| License | Apache 2.0 |
| Maintaining Organization | OSEHRA |
| Last Activity | Recent |
| GT.M Compatibility | Yes |
| Caché Compatibility | Yes |
| Programming Language | C# |
| VistA Awareness | Yes |

MDWS, short for Medical Domain Web Services, is a suite of web services that exposes medical domain functionality. MDWS has the ability to expose any VistA remote procedure call as a web service.

**4. Forums, Lists, and Blogs**

HardHats Mailing List - <https://groups.google.com/forum/#!forum/Hardhats>

OSEHRA Blogs and Discussions - <http://www.osehra.org/content/blogs>

The EWD Files Blog - <http://robtweed.wordpress.com>

Vistapedia - <http://www.vistapedia.com/index.php/Main_Page>

**5. Frequently Asked Questions**

**1. What is an Electronic Health Record?**

An electronic health record (EHR) (also known as an electronic medical record) is a digital version of a paper chart that contains all of a patient’s medical history from one practice. An EMR is mostly used by providers for diagnosis and treatment.

Reference: <http://www.healthit.gov/providers-professionals/electronic-medical-records-emr>.

**2. What is VistA?**

The Veterans Health Information Systems and Technology Architecture (VistA) is an enterprise-wide information system built around an Electronic Health Record (EHR), used throughout the United States Department of Veterans Affairs (VA) medical system, known as the Veterans Health Administration (VHA).

Reference: <http://en.wikipedia.org/wiki/VistA>.

**3. What is MUMPS?**

MUMPS (Massachusetts General Hospital Utility Multi-Programming System, later: 'Multi-User Multi-Programming System') or alternatively M, is a general-purpose computer programming language that provides ACID (Atomic, Consistent, Isolated, and Durable) transaction processing. Its most unique and differentiating feature is its "built-in" database, enabling high-level access to disk storage using simple symbolic program variables and subscripted arrays, similar to the variables used by most languages to access main memory.

Reference: <http://en.wikipedia.org/wiki/MUMPS>.

**4. Why is it important to engage with VistA using open source/mainstream technologies?**

There are a significantly greater number of developers that are familiar with mainstream technologies (like JavaScript) in comparison to MUMPS. By wrapping VistA in mainstream technologies, it opens up opportunities for a broad community of developers that can build innovative capabilities while leveraging the power of VistA. Also, by wrapping VistA in “business logic” instead of exposing users directly to the underlying MUMPS technology, it reduces the amount of expertise that developers must have with VistA before getting involved.

**6. Invitation to Contribute**

The VistA Application Developer Documentation is intended to evolve over time with community input and introduction of new technologies and capabilities. As such, this document will be published to a public wiki and we welcome your contributions to the documentation.