VistA M-to-M BROKER

Patch XWB*1.1*28

August 2002

Updated: Patch XWB*1.1*34

October 2005

Draft

Department of Veterans Affairs
VistA Health Systems Design & Development (HSD&D)
Infrastructure & Security Services (ISS)

Revision History

Document History

The following table displays the revision history for this document.

Date	Description	Author
August 2002	Initial M2M Broker documentation, Patch XWB*1.1*28 software release.	Susan Strack, Oakland OIFO, Raul Mendoza, Oakland OIFO
October 2002	Updated documentation to include the entry point that must be included in the COM file to connect the listener to the M-to-M Broker.	Susan Strack, Oakland OIFO, Raul Mendoza, Oakland OIFO
October 2005	Updated documentation in support of Patch XWB*1.1*34 software release.	Susan Strack, Oakland OIFO, Raul Mendoza, Oakland OIFO

Patch History

For the current patch history related to this software, please refer to the Patch Module (i.e., Patch User Menu [A1AE USER]) on FORUM.

Contents

Revision History iii

Tables vii

Orientation ix

Introduction xiii

Chapter 1: Software Dependencies 1-1

Chapter 2: New Listener: Create a TCP/IP Service for VMS/Caché 2-1

Chapter 3: New Message Structure 3-1

Chapter 4: Security Features 4-1

Chapter 5: Use Case—How to Run an M-to-M Broker RPC 5-1

Chapter 6: VistA M-to-M Broker APIs 6-1

Chapter 7: Technical Information 7-1

- Implementation and Maintenance 7-1
- Software Dependencies 7-1
- Routines 7-2
- Options 7-2
- Archiving and Purging 7-3
- Callable Routines 7-3
- External Interfaces 7-4
- External Relations 7-4
- Internal Relations 7-4
- Software Product Security 7-5

Glossary Glossary-1

Appendix A: Error Messages .... Appendix A-1

Index Index-1

Tables

Table i-1: Documentation symbol descriptions ix

Table 1-1: Software Dependency 1-1

Table 4-1: Security Tasks 4-2

Table 6-1: API—$$CONNECT^XWBM2MC input parameters 6-2

Table 6-2: API—$$CONNECT^XWBM2MC output 6-2

Table 6-3: API—$$SETCONTX^XWBM2MC input parameter 6-4

Table 6-4: API—$$SETCONTX^XWBM2MC output 6-4

Table 6-5: API—$$GETDIV^XWBM2MC output 6-5

Table 6-6: API—$$SETDIV^XWBM2MC input parameter 6-7

Table 6-7: API—$$SETDIV^XWBM2MC output 6-7

Table 6-8: API—$$PARAM^XWBM2MC input parameters 6-9

Table 6-9: API—$$PARAM^XWBM2MC output 6-9

Table 6-10: API—$$CALLRPC^XWBM2MC input parameters 6-11

Table 6-11: API—$$CALLRPC^XWBM2MC output 6-11

Table 6-12: API—$$CLOSE^XWBM2MC output 6-13

Table 6-13: API—$$GETCONTX^XWBM2MC input parameter 6-14

Table 6-14: API—$$GETCONTX^XWBM2MC output 6-14

Table 7-1: Callable entry points exported with the M-to-M Broker 7-4

Table A-1: Error message—Control Character Found Appendix A-1

Table A-2: Error message—Could not obtain list of valid divisions for current user Appendix A-1

Table A-3: Error message—Could not open connection Appendix A-1

Table A-4: Error message—Could not Set active Division for current user Appendix A-1

Table A-5: Error message—Invalid user, no DUZ returned Appendix A-2

Table A-6: Error message—RPC could not be processed Appendix A-2

Table A-7: Error message—There is no connection Appendix A-2

Table A-8: Error message—XUS AV CODE RPC failed Appendix A-2

Table A-9: Error message—XUS SIGNON SETUP RPC failed Appendix A-3

Table A-10: Error message—Remote Procedure Unknown Appendix A-3

Orientation

Throughout this manual, advice and instructions are offered regarding the use of the M-to-M Broker and the functionality it provides VistA.

How to Use This Manual

This manual uses several methods to highlight different aspects of the material:

- Various symbols are used throughout the documentation to alert the reader to special information. The following table gives a description of each of these symbols:

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

Table i-1: Documentation symbol descriptions

- Descriptive text is presented in a proportional font (as represented by this font).
- "Snapshots" of computer online displays (i.e., character based interface screen captures/dialogs) and computer source code are shown in a *non*-proportional font and enclosed within a box.
- All uppercase is reserved for the representation of M code, variable names, or the formal name of options, field and file names, and security keys (e.g., the XUPROGMODE key).
- Conventions for displaying TEST data in this document are as follows:
  - The first three digits (prefix) of any Social Security Numbers (SSN) will begin with either "000" or "666".
  - Patient and user names will be formatted as follows: [Application Name]PATIENT,[N] and [Application Name]USER,[N] respectively, where "Application Name" is defined in the Approved Application Abbreviations document, located on the [Web site] and where "N" represents the first name as a number spelled out and incremented with each new entry.

ℹ The list of Approved Application Abbreviations can be found at the following Web site:

http://vista.med.va.gov/iss/strategic_docs.asp#sop

Presentation Structure

This documentation is intended for use in conjunction with the M-to-M Broker, Patch XWB*1.1*34. It is divided into the following chapters and appendix:

- "Chapter 1: Software Dependencies" details the software and hardware setup requirements needed by the M-to-M Broker.
- "Chapter 2: New Listener: Create a TCP/IP Service for VMS/Caché" introduces a new listener (TCP/IP Service) for Caché sites running on VMS operating systems.

For information on setting up and starting the TCP/IP Service, see the *TCP/IP Supplement*, *Patch XWB*1.1*35* on the VistA Documentation Library (VDL) at:

http://www.va.gov/vdl/Infrastructure.asp?appID=23

- "Chapter 3: New Message Structure" details the M-to-M Broker input and output message structures wrapped in an Extensible Markup Language (XML) format.
- "Chapter 4: Security Features" details the robust security features of the M-to-M Broker.
- "Chapter 5: Use Case—How to Run an M-to-M Broker RPC" describes a scenario using the M-to-M Broker APIs to create and run a Remote Procedure Call (RPC).
- "Chapter 6: VistA M-to-M Broker APIs" details the Application Program Interfaces (API) exported with the M-to-M Broker.
- "Chapter 7: Technical Information" provides technical information specific to the implementation and maintenance of the M-to-M Broker, as well as routines, options, callable routines, software product security, etc. (i.e., the product Technical Manual).
- Appendix A: Error Messages describes the possible error messages encountered during M-to-M Broker Client/Server processing within the VistA environment.

Assumptions About the Reader

This manual is written with the assumption that the reader is familiar with the following:

- VistA computing environment (e.g., Kernel Installation and Distribution System [KIDS])
- VA FileMan data structures and terminology
- M programming language

No attempt is made to explain how the overall VistA programming system is integrated and maintained. Such methods and procedures are documented elsewhere. We suggest you look at the various VA home pages on the World Wide Web for a general orientation to VistA. For example, go to the Health System Design & Development (HSD&D) Home Page at the following Web address:

http://vaww.vista.med.va.gov/.

This manual does provide, however, an explanation of the M-to-M Broker, describing how it can be used in an M based server-to-server environment.

Reference Materials

Readers who wish to learn more about the M-to-M Broker should consult the following:

- Vista M-to-M Broker, Patch XWB*1.1*34 documentation (written for programmers) is made available online in Adobe Acrobat Portable Document (PDF) Format and can be found on the VistA Documentation Library (VDL) at the following Web address:

  http://vaww.vista.med.va.gov/vdl/Infrastructure.asp#App128 .

  VistA M-to-M Broker documentation and software can also be downloaded from the Enterprise VistA Support (EVS) anonymous directories:

  - ➢ **Preferred Method: download.vista.med.va.gov**
  - ➢ Albany OIFO: ftp.fo-albany.med.va.gov
  - ➢ Hines OIFO: ftp.fo-hines.med.va.gov
  - ➢ Salt Lake City OIFO: ftp.fo-slc.med.va.gov

  This method transmits the files from the first available FTP server.

- M-to-M Broker Installation Instructions can be found in the XWB*1.1*34 patch description, located in the Patch Module (i.e., Patch User Menu [A1AE USER]) on FOURM.
- TCP/IP Supplement, Patch XWB*1.1*35 can be found on the VistA Documentation Library (VDL) at the following Web address:

  http://www.va.gov/vdl/Infrastructure.asp?appID=23

  Software and installation instructions can be found in the patch description for the same, located in the Patch Module (i.e., Patch User Menu [A1AE USER]) on FOURM.
- M-to-M Broker Home Page is located at the following Web address:

  http://vaww.vista.med.va.gov/m2m_broker/index.asp.

DISCLAIMER: The appearance of any external hyperlink references in this manual does not constitute endorsement by the Department of Veterans Affairs (VA) of this Web site or the information, products, or services contained therein. The VA does not exercise any editorial control over the information you may find at these locations. Such links are provided and are consistent with the stated purpose of this VA Intranet Service.

Introduction

This documentation is intended for use in conjunction with the VistA M-to-M Broker, Patch XWB*1.1*34. It provides descriptive information and instructions on the use of the M-to-M Broker software.

The audience for this documentation is all key stakeholders. The primary stakeholders are the VistA Infrastructure & Security Services (ISS) and VistA Imaging Service. Additional stakeholders include all VA facilities that utilize VistA Imaging Package, CIO Technical Services, and Veterans Health Information Systems and Technology Architecture (VistA) sites.

Overview

The VistA M-to-M Broker is a new implementation of the RPC Broker offering Client/Server functionality resident solely within a VistA *non*-Graphical User Interface (GUI) environment. It enables the exchange of VistA M-based data and business rules between two VistA M servers, where both servers reside on local and/or remote VistA systems:

- The requesting server functions in the capacity of a Client.
- The server receiving that request functions in the capacity of a Server.

The Client/Server roles of each server can vary depending on what point in time each VistA M server is making the request for data from its counterpart VistA M server.

All M-to-M Broker client and server routines are packaged in one KIDS build, Patch XWB*1.1*34, which will need to be installed on all VistA systems required for M-to-M Broker processing.

Scope

The M-to-M Broker provides a new implementation of the RPC Broker enabling the exchange of VistA M-based data and business rules between two VistA M servers, where both servers reside on the same, or on different VistA M systems.

For the VistA Imaging Digital Imaging and Communication in Medicine (DICOM) Gateway, the M applications on separate VistA systems will be converted to use this new M-to-M Broker software to communicate to the main VistA Hospital Information System (HIS). This eliminates the need for Distributed Data Processing (DDP).

Background

VistA Imaging requested the development of the M-to-M Broker to be used to communicate between the M client on the VistA Imaging DICOM Gateway and the M server on the main HIS.

The VistA Imaging DICOM Gateway architecture uses M software on a workstation to create associations between newly acquired images and computerized patient records. Before the development of the M-to-M Broker, the gateway system communicated with the main Hospital Information System using the DDP protocol, stored the acquired images on Microsoft Operating System (NT) file servers, and set database entries to reference them.

Problems with DDP

- Causes database inconsistencies
- Lacks security
- Bound to MAC addresses
- Responds slow on a busy Hospital Information System and/or network
- Runs very slowly in a Wide Area Network (WAN) environment because of inherent network latencies

Because of the database inconsistency problem, incidents of matching images to the wrong patient occurred at one particular site.

DDP provides no security. M-to-M Broker uses many of the robust security features implemented by the VistA RPC Broker and Kernel software. These security features are transparent to the end-user and without additional impact on Information Resources Management (IRM).

About the Remote Procedure Call (RPC) Broker

The RPC Broker (also referred to as "Broker") is a Client/Server system within VA's Veterans Health Information Systems and Technology Architecture (VistA) environment. It establishes a common and consistent foundation for Client/Server applications being written as part of VistA. It enables client applications to communicate and exchange data with M Servers.

The RPC Broker is a bridge connecting the client application front-end on the workstation (e.g., Delphi GUI applications) to the VistA M-based data and business rules on the server. It links one part of a program running on a workstation to its counterpart on the server.

For information on the RPC Broker, documentation is made available online in Adobe Acrobat Portable Document Format (PDF) at the following Web address: http://vaww.vista.med.va.gov/vdl/Infrastructure.asp#App23 .

Chapter 1: Software Dependencies

M-to-M Broker requires that both Test and Production accounts exist in a standard VistA operating environment in order to function correctly. The accounts must contain the *fully* patched versions of the following software:

- Kernel V. 8.0
- Kernel Toolkit V. 7.3
- VistA Extensible Markup Language (XML) Parser, Patch XT*7.3*58
- RPC Broker V. 1.1
- VA FileMan V. 22.0

In addition to a standard VistA operating environment, the following patch must be installed prior to Patch XWB*1.1*34:

VistA Software and Version	**Associated Patch Designation(s)**	**Brief Patch Description**
RPC Broker V. 1.1	XWB*1.1*35	NON-callback server.

Table 1-1: Software Dependency

Chapter 2: New Listener: Create a TCP/IP Service for VMS/Caché

In order for Caché sites running on VMS operating systems to utilize the M-to-M Broker, it is necessary to setup and start a TCP/IP Service. (If your site has already set up a TCP/IP service resulting from Patches XWB*1.1*28 and XWB*1.1*34, use that service.) This Service is a new listener for VMS/Caché, which establishes a connection using TCP/IP. It is always listening; it never shuts down.

For complete information on setting up and starting the TCP/IP Service , see the *TCP/IP Supplement, Patch XWB*1.1*35* on the VistA Documentation Library (VDL) at:

http://www.va.gov/vdl/Infrastructure.asp?appID=23

Chapter 3: New Message Structure

M-to-M Broker generates requests and results (input and output) messages, which are wrapped in an Extensible Markup Language (XML) format. The VistA M requesting server makes a connection using the TCP/IP service, triggering the following consecutive actions:

1. The VistA Extensible Markup Language (XML) Parser (Kernel Toolkit Patch XT*7.3*58) parses out the name of the remote procedure call and, when included, its parameters.
2. The M-to-M Broker looks up the remote procedure call in the REMOTE PROCEDURE file (#8994) and executes the RPC using the passed parameters.
3. The server processes the request and returns the results of the operation.
    - If the operation is a query, the result is a set of records that satisfy that query.
    - If the operation files data on the server, or if there is no need to return any data, notification of the successful operation is returned to the requesting server.

For information on VistA Extensible Markup Language (XML) Parser, Kernel Toolkit Patch XT*7.3*58, please refer to the "VistA Extensible Markup Language (XML) Parser Technical and User Documentation," located at:

http://www.va.gov/vdl/Infrastructure.asp?appID=137 .

Create Your Own Custom RPCs

You can create your own custom RPCs to perform actions on and retrieve data from the VistA M server.

For information on how to create custom RPCs, refer to the *Getting Started With The Broker Development Kit (BDK)* manual in the chapter titled "Remote Procedure Calls (RPCs)." It is made available online in Adobe Acrobat Portable Document (PDF) format at the following Web address:

http://www.va.gov/vdl/Infrastructure.asp?appID=23 .

Everything in this chapter is applicable to M-to-M Broker processing, with the exception of the Delphi-specific section titled "How to Execute an RPC from a Client Application."

Chapter 4: Security Features

The M-to-M Broker implements robust security without additional impact on IRM. Security with the M-to-M Broker is a four-part process:

Step	**Description**
1.	Client workstations must have a valid connection to the appropriate listener: It is encouraged that VMS/Caché sites use the TCP/IP service.
2.	Users must have valid Access and Verify codes.
3.	Remote procedure calls (RPC) must be registered and valid for the application being executed.
4	Authenticated VistA users must be assigned to the appropriate B-type option, verifying permission to run the RPCs related to the VistA application they are using.

Validation of RPCs

M-to-M Broker security allows any RPC to run when it is properly registered to the VistA Client/Server application. The Broker on the server, along with Kernel's Menu Manager determines which application a user is currently running. Menu Manager determines if a user is allowed to run this application or option by the following process:

Step	**Description**
1.	An RPC is sent by a client application and is received by the M-to-M Broker on the server.
2.	The M-to-M Broker verifies that the RPC is "registered" to the application that the user is currently running, *prior* to executing the RPC. The application being run is designated by a B-type option in the OPTION file (#19). The application must specify the option and that option *must* be in a user's menu tree. For more information on registering an RPC to a package, please refer to the "RPC Security: How to Register An RPC" topic in the *RPC Broker Getting Started with the Broker Development Kit (BDK) manual*.
3.	Menu Manager checks if the RPC is registered for this package option. If not properly registered, Menu Manager will return a message explaining why the RPC is not allowed.
4.	If registered, the M-to-M Broker executes the RPC on the server. Otherwise, it is rejected.

Sample Security Procedures

The security steps each client follows and the intermediate Client/Server security processes are described in the following example:

Step	Description
1.	The user starts a VistA program on the client.
2.	The user signs onto the server through the VistA sign-on dialog on the client using their Access and Verify codes, invoking the Kernel sign on process.
3.	The Menu Manager on the server verifies the user is allowed access to the B-type option requested by CPRS.
4.	The Menu Manager on the server verifies the option is a "Client/Server"-type option and the requested RPC resides in that option's RPC multiple.
5.	If all of the previous steps complete successfully, the application RPC is launched.

Security Features Tasks Summary

The following table summarizes required security tasks:

Security Task	Completed By
Verify valid connection request	RPC Broker
Verify valid user	Kernel Signon
Verify user is authorized to run this package	RPC Broker & Menu Manager
Verify an RPC is registered to an application	RPC Broker & Menu Manager
Application—RPC Registration	Kernel Installation and Distribution System (KIDS)

Table 4-1: Security Tasks

Chapter 5: Use Case—How to Run an M-to-M Broker RPC

This section attempts to explain the integrated use of the M-to-M Broker APIs to enable the exchange of VistA M-based data and business rules between two VistA M servers, where both servers reside on local and/or remote VistA systems. The M-to-M Broker uses these APIs to run an RPC, detailed as follows:

Step	Description
1.	A connection is established between two VistA servers.
2.	The application context is established, which is the environment necessary to run the associated RPCs in VistA.
3.	If a user has multi-divisional access, the selection of a division needs to be established.
4.	An RPC structure is built and makes the call to the RPC on the server.
5.	The connection is closed between that particular instance of the requesting and receiving VistA servers, and any necessary cleanup is performed.
6.	Instructions are provided for VistA application developers how to transmit control characters through M-to-M Broker RPCs.

The M-to-M Broker APIs are documented in detail in the section titled: "VistA M-to-M Broker APIs" in this manual.

Establish the Connection to the VistA M Sever

Use the $$CONNECT^XWBM2MC API to establish the initial connection to the VistA M server via the IP address and the port number for that listener. (Port 4800 is reserved in your main Production account for M-to-M Broker.) The three input parameters used by this API are:

1. PORT—This is the port number where the connection to the VistA M server is established and running.
2. IP—This is the IP address where the connection to the VistA M server is established and running.
3. AV—This parameter contains the VistA Access and Verify codes to sign onto the system. The Access and Verify codes passed in the AV input parameter are used to authenticate that a valid VistA user is connecting to the server. They provide a critical element of security offered to VistA by the M-to-M Broker.

This API is an extrinsic function returning a 1 or 0 indicating success or failure to connect to the VistA server. In addition, the ^TMP global will be updated to 1 or 0, as shown below:

^TMP("XWBM2M",$J,"CONNECTED") = 1 (successful connection established)

^TMP("XWBM2M",$J,"CONNECTED") = 0 (connection failed)

The contents of the ^TMP global can be used by the developer as an internal reference for the application.

Any errors encountered during the processing of the M-to-M Broker APIs will be written to this ^TMP global: ^TMP("XWBM2ME",$J,"ERROR"). See "Appendix A: Error Messages" in this manual for details.

Set Up the Environment to Run the RPCs in VistA

Now that you have a connection to the VistA server, use the $$SETCONTX^XWBM2MC API to set up the application context for the necessary environment to run the M-to-M Broker RPCs in VistA.

What is a VistA Application Context?

Application context, as referred to in VistA, is a B-type option located in the OPTION file (#19). This option is assigned to an authenticated VistA user. It verifies that the user has permission to run RPCs related to specific VistA applications as defined by the application developers. The application context has to be set for every VistA application that uses the M-to-M Broker. The associated context name (B-type option) has to be assigned to each user who is using that VistA application. This is another critical element of security offered to VistA by the M-to-M Broker.

The $$CONNECT^XWBM2MC API has already authenticated the user to VistA. Next, the $$SETCONTX^XWBM2MC API sets the application context for that user verifying access to the B-type option associated with the RPCs used by the VistA application.

For example, XWB BROKER EXAMPLE is the name of an application context, which is a B-type option located in the OPTION file (#19). This option has several RPCs associated with it. In order for users to run RPCs linked to this option, XWB BROKER EXAMPLE needs to be assigned to their secondary menu.

The $$SETCONTX^XWBM2MC API uses only one input parameter named CONTXNA, which contains the B-type option name identifying the application context to be set for the application. This API uses CONTXNA to do a lookup on the OPTION file (#19). Once the user is verified as having access to this B-type option name contained in CONTXNA, $$SETCONTX^XWBM2MC sends back the number 1, indicating that the application context has been successfully set. If unsuccessful, it sends back the number 0. Once verified as having access, the user is then able to run any RPCs associated with that B-type option.

If this function is successful, the application context name is stored in the ^TMP global:

```
^TMP("XWBM2M",$J,"CONTEXT")
```

When is it Not Necessary to Set the Application Context?

With respect to application context in VistA, if you have programmer access (the @ sign) as a developer, all security is bypassed. Hence, you would not need to use the $$SETCONTX^XWBM2MC API.

Switching Between Application Contexts

The $$GETCONTX^XWBM2MC API allows you to switch between application contexts so users can run RPCs linked to different B-type options. It returns the current application context so that a new context may be established, thereby restoring the previous application context prior to switching to the new one. Developers can use this API to keep track of multiple application contexts as required.

The M-to-M Broker provides the tools to set and retrieve the current application context.

Applications must keep track of where the different application contexts are saved in order to access them when they are needed.

Obtain and Set the Division for the Current User or Logon Session

The $$GETDIV^XWBM2MC API obtains a list of valid divisions for a particular user or logon session. The IEN, station name, and station number are returned for each valid division. If a user has only 1 division, then XWBDIVG(1) is equal to the value 0 (zero) because Kernel automatically assigns that division as the default. Use IEN to set division in $$SETDIV.

Next, the $$SETDIV^XWBM2MC API sets the active division for a particular user or logon session. If only one division is associated with a logon session (e.g., XWBDIVG(1)=0), Kernel automatically assigns that division as a default.

When two or more facilities are integrated, the legacy facilities become the divisions of the primary facility.

Build and Request an RPC to Run

The next step is to build the RPC structure and make the call to the RPC. This can include one or both of the following APIs:

- $$PARAM^XWBM2MC builds the PARAM Data Structure.
- $$CALLRPC^XWBM2MC builds the Remote Procedure Call data structure, then makes the call to the RPC on the server.

Using $$PARAM^XWBM2MC With $$CALLRPC^XWBM2MC

Application developers have to know ahead of time which RPCs to call because the $$CALLRPC^XWBM2MC API requires that they include the name of the RPC as the input parameter RPCNAM. Typically, the developer who sets up an RPC beforehand knows if that RPC requires any data to run. RPCs don't require input data to run. If an application requires an RPC send data, the $$PARAM^XWBM2MC API must be used to set up an array with the necessary data.

The $$PARAM^XWBM2MC API requires the following two input parameters:

1. PARAMNUM—This input parameter contains a number with which to associate the VALUE and TYPE to the RPC. The value of PARAMNUM should start with the number 1.

   - VALUE contains the data that the RPC needs to run
   - TYPE contains the data type, which can be a string, reference, or an array

2. ROOT—This input parameter is a value passed by reference. ROOT contains the VALUE and TYPE necessary to run the RPC.

Both $$PARAM^XWBM2MC and $$CALLRPC^XWBM2MC are extrinsic functions returning a success/fail indicator of 1 or 0, respectively.

Using $$CALLRPC^XWBM2MC as a Standalone API

The $$CALLRPC^XWBM2MC API can be used standalone, without $$PARAM^XWBM2MC to set up an array with the data. This API builds the RPC data structure and then makes the call to the RPC on the server. The request message is transported in XML and is parsed by the VistA Extensible Markup Language (XML) Parser, introduced in Kernel Toolkit Patch XT*7.3*58.

The $$CALLRPC^XWBM2MC API requires the following three input parameters:

1. RPCNAM is the name of the RPC called on the server.
2. RES (also used as an output parameter) contains the result when the RPC returns data. If the value of RES is null, the results are stored in ^TMP("XWBM2MRPC",$J,"RESULTS").
3. CLRPARMS clears (or kills) the parameters array after the RPC has been processed based on the following return value:
   - 1—parameter array is killed
   - 0—parameter array is *not* killed
   - null—parameter array is killed (default value)

Close the VistA Server Connection

Use the $$CLOSE^XWBM2MC API to close the connection between that particular instance of the requesting and receiving VistA servers, then perform any necessary cleanup. This API uses an internal

RPC to make one last call to the server so it can shut down gracefully and does some cleanup work on the VistA server.

In addition to this extrinsic function returning a 1 or 0 indicating success or failure to close the VistA server-to-server connection, the ^TMP global will be updated to 0, as shown below:

```
^TMP("XWBM2M",$J,"CONNECTED") = 0
```

This can be used as an internal reference for the application.

When Do I Leave the Connection Open?

This is a straightforward connection: run one RPC, and then close the connection. Realistically, however, your application may require the connection stay open and run multiple RPCs. Once you make a connection to the VistA server, it can stay open. Your application may require a loop to interchangeably call the M-to-M Broker APIs. You can change the application context, make multiple calls to RPCs, or depending on your applications requirements, you can keep the connection open and run RPCs continuously until your application flags it to be closed.

Control Character Handling

VistA application developers needing to transmit control characters through M-to-M Broker RPCs must make code allowances to translate the control characters to their ASCII values. The translated ASCII values are then passed in the M-to-M Broker.

Chapter 6: VistA M-to-M Broker APIs

M-to-M Broker provides a new implementation of the RPC Broker offering Client/Server functionality resident solely within a VistA *non*-Graphical User Interface (GUI) environment. This chapter provides detailed information on the APIs exported with the M-to-M Broker. These APIs are open for use by any VistA application as defined by the Integration Agreement (IA) introduced by this release. They have been recorded as a Supported Reference in the IA database on FORUM. It is not required that VistA packages request an IA to use them.

M-to-M Broker APIs

This section lists the APIs exported with the M-to-M Broker in order of operation by entry point, providing a description of their:

- use
- format
- input parameters
- output
- usage

$$CONNECT^XWBM2MC—M Client/Server Connection

This API establishes the initial connection to the VistA M server. It is an extrinsic function that returns a success/fail indicator of 1 or 0, respectively.

Format:

$$CONNECT^XWBM2MC(PORT,IP,AV)

Input/Output:

Input	Description
PORT	(Required) Port number where the connection to the VistA M server is established and running. (Port 4800 is reserved in your main Production account for M-to-M Broker.)
IP	(Required) IP address where the connection to the VistA M server is established and running.
AV	(Required) Access and Verify codes to sign onto the VistA system.

Table 6-1: API—$$CONNECT^XWBM2MC input parameters

Output	Description
1	The initial M Client/Server connection was successfully established.
0	The initial M Client/Server connection failed.

Table 6-2: API—$$CONNECT^XWBM2MC output

Details:

In addition to this function returning a 1 or 0 indicating success or failure to make an M Client/Server connection, a 1 or 0 will also be written to the ^TMP global, shown below:

^TMP("XWBM2M",$J,"CONNECTED") = 1 (successful connection established)

^TMP("XWBM2M",$J,"CONNECTED") = 0 (connection failed)

The value written to the ^TMP global can be used as an internal reference for the application.

The following are error messages, which, if encountered during processing, are written to the ^TMP global shown below:

^TMP("XWBM2ME",$J,"ERROR",”CONNECT”) = Could not open connection

^TMP("XWBM2ME",$J,"ERROR",”SIGNON”) = XUS SIGNON SETUP RPC failed

^TMP("XWBM2ME",$J,"ERROR",”SIGNON”) = XUS AV CODE RPC failed

^TMP("XWBM2ME",$J,"ERROR",”SIGNON”) = Invalid user, no DUZ returned

See “Appendix A: Error Messages” for more information on error messages associated with the M-to-M Broker.

Example:

SET CONNECT=$$CONNECT^XWBM2MC(4800, "10.9.8.7","smith;password")

If successful:

CONNECT=1

^TMP("XWBM2M",$J,"CONNECTED") = 1

If *not* successful:

CONNECT=0

$$SETCONTX^XWBM2MC—Set Application Context

This API sets the context, creating the necessary environment to run the RPCs. It is an extrinsic function that returns a success/fail indicator of 1 or 0, respectively.

Format:

$$SETCONTX^XWBM2MC(CONTXNA)

Input/Output:

Input	Description
CONTXNA	(Required) Desired application context name.

Table 6-3: API—$$SETCONTX^XWBM2MC input parameter

Output	Description
1	Application context was successfully set.
0	Application context failed to be set.

Table 6-4: API—$$SETCONTX^XWBM2MC output

Details:

If this function is successful, the application context name is stored in the ^TMP global shown below:

```
^TMP("XWBM2M",$J,"CONTEXT")
```

Example:

```
SET CONTEXT=$$SETCONTX^XWBM2MC("XWB BROKER EXAMPLE")
```

If successful:

```
CONTEXT=1
^TMP("XWBM2M",$J,"CONTEXT") = XWB BROKER EXAMPLE
```

If *not* successful:

```
CONTEXT=0
```

$$GETDIV^XWBM2MC—Get Division for Current User or Logon Session

This API obtains a list of valid divisions for a particular user or logon session. It is an extrinsic function that returns a success/fail indicator of 1 or 0, respectively.

When two or more facilities are integrated, the legacy facilities become the divisions of the primary facility.

Format:

$$GETDIV^XWBM2MC(XWBDIVG)

Input/Output:

Output	Description
XWBDIVG	(Required) This variable stores the results of the call. It holds division information associated with the logon session detailed as follows: XWBDIVG(1) = number of divisions contained in Subscript 1 XWBDIVG(#) = 'IEN;station name;station#' contained in Subscript #, delineated with ";". Consecutive subscripts (e.g., XWBDIVG(2), XWBDIVG(3), etc.) return the individual strings containing the IEN, station name, and station number of each division. If a user has only 1 division, then XWBDIVG(1)=0 because Kernel automatically assigns that division as the default. Use IEN to set division in $$SETDIV.
1	Application successfully obtained the division for this logon session.
0	Application failed to obtain the division for this logon session.

Table 6-5: API—$$GETDIV^XWBM2MC output

The following are error messages, which, if encountered during processing, are written to the ^TMP global shown below:

^TMP("XWBM2ME",$J,"ERROR",”GETDIV”) = Could not obtain list of valid divisions for current user

See “Appendix A: Error Messages” for more information on error messages associated with the M-to-M Broker.

Example:

```
SET DIVFLAG=$$GETDIV^XWBM2MC("DIVISIONS")
```

If successful:

```
DIVFLAG=1
DIVISIONS(1)=3
DIVISIONS(2)=1^San Francisco^662
DIVISIONS(3)=2^New York^790
DIVISIONS(4)=3^San Diego^664
```

If *not* successful:

```
DIVFLAG=0
```

$$SETDIV^XWBM2MC—Set Division for Current User or Logon Session

This API sets the active division for a particular user or logon session. It is an extrinsic function that returns a success/fail indicator of 1 or 0, respectively.

Format:

$$SETDIV^XWBM2MC(XWBDIVS)

Input/Output:

Input	Description
XWBDIVS	(Required) The division IEN number (See the **XWBDIVG** parameter in the $$GETDIV^XWBM2MC—Get Division for Current User or Logon Session API documentation.) is passed in to set the division. If only 1 division is associated with a logon session, (then XWBDIVG(1)=0) Kernel automatically assigns that division as a default.

Table 6-6: API—$$SETDIV^XWBM2MC input parameter

Output	Description
1	Application successfully set the active division for this logon session.
0	Application failed to set the active division for this logon session.

Table 6-7: API—$$SETDIV^XWBM2MC output

The following are error messages, which, if encountered during processing, are written to the ^TMP global shown below:

^TMP("XWBM2ME",$J,"ERROR",”SETDIV”) = Could not Set active Division for current user

See “Appendix A: Error Messages” for more information on error messages associated with the M-to-M Broker.

Example:

SET DIVISION=$$SETDIV^XWBM2MC(XWBDIVS)

Where XWBDIVS=1 for San Francisco

If successful:

DIVISION=1

If *not* successful:

```
DIVISION=0
```

$$PARAM^XWBM2MC—Build the PARAM Data Structure

This API sets up the PARAM data structure necessary to run the RPCs. It is an extrinsic function that returns a success/fail indicator of 1 or 0, respectively.

Format:

$$PARAM^XWBM2MC(PARAMNUM,ROOT)

Input/Output:

Input	Description
PARAMNUM	(Required) This is a number to associate the VALUE and TYPE with a parameter to the RPC. It should start with the number 1.
ROOT	(Required) Value passed by reference. You can use this variable to obtain the values. The ROOT contains the VALUE and TYPE necessary to run the RPC. VALUE: The data that the RPC needs to run. TYPE: The datatype (string, reference, array) of the data.

Table 6-8: API—$$PARAM^XWBM2MC input parameters

Output	Definition
1	The PARAM data structure was successfully created.
0	The PARAM data structure failed to be created.

Table 6-9: API—$$PARAM^XWBM2MC output

Example 1:

SET X=$$PARAM^XWBM2MC(PARAMNUM,ROOT)

If successful:

X=1

Where ROOT could be $NA(^TMP("IMG",$J)) and the values under ROOT could look like:

^TMP("IMG",$J,"TYPE")="STRING"

^TMP("IMG",$J,"VALUE")="XWBTEST"

If *not* successful:

X=0

Example 2:

SET X=$$PARAM^XWBM2MC(1,$NA(^TMP("IMG",$J))

If successful:

X=1

Where ROOT is "^TMP("IMG",$J)" and the values under ROOT could look like:

```
^TMP("IMG",$J,"TYPE")="ARRAY"
^TMP("IMG",$J,"VALUE","M2MPROGRAMMER,ONE")="PROGRAMMER"
^TMP("IMG",$J,"VALUE","M2MTECHWRITER,ONE")="TECH WRITTER"
```

If *not* successful:

X=0

$$CALLRPC^XWBM2MC—Build the Remote Procedure Data Structure

This API builds the Remote Procedure Call (RPC) data structure and makes the call to the RPC on the server. The request message is transported in XML and is parsed by the VistA Extensible Markup Language (XML) Parser, introduced in Kernel Toolkit Patch XT*7.3*58.

This API is an extrinsic function that returns a success/fail indicator of 1 or 0, respectively.

Format:

$$CALLRPC^XWBM2MC(RPCNAM,RES,CLRPARMS)

Input/Output:

Input	Description
RPCNAM	(Required) This is the name of the RPC called on the server.
RES	This is where the result is placed. If the value of RES is null, the results will be placed in: ^TMP("XWBM2MRPC",$J,"RESULTS").
CLRPARMS	After the RPC has been processed, CLRPARMS clears (kills) the parameters array based on the return value: If CLRPARMS = 1, the parameter array is killed. If CLRPARMS = 0, parameter array is *not* killed. If CLRPARMS = null, the default is to kill the parameter array.

Table 6-10: API—$$CALLRPC^XWBM2MC input parameters

Output	Definition
RES	Stores results of the RPC.
^TMP("XWBM2MRPC",$J,"RESULTS")	If the value of RES is null, the results will be placed in this global.
1	Call to RPC was successful.
0	Call to RPC failed.

Table 6-11: API—$$CALLRPC^XWBM2MC output

Details:

The following are error messages, which, if encountered during processing, are written to the ^TMP global shown below:

^TMP(“XWBM2ME”,$J,”ERROR”,”CALLRPC”) = There is no connection

^TMP(“XWBM2ME”,$J,”ERROR”,”CALLRPC”) = RPC could not be processed

^TMP(“XWBM2ME”,$J,”ERROR”,”CALLRPC”) = Control Character Found

See “Appendix A: Error Messages” for more information on error messages associated with the M-to-M Broker.

Example:

SET CALL=$$CALLRPC^XWBM2MC("XWB EXAMPLE ECHO STRING", "REQ",1)

If successful:

CALL=1

REQ(1) = XWBTEST

If *not* successful:

CALL=0

$$CLOSE^XWBM2MC—Close Connection

This API closes the connection between that particular instance of the requesting and receiving VistA M servers, and performs any necessary cleanup. It is an extrinsic function that returns a success/fail indicator of 1 or 0, respectively.

Format:

$$CLOSE^XWBM2MC

Output:

Output	Definition
RES	Stores results of the RPC.
^TMP("XWBM2MRPC",$J,"RESULTS")	If the value of RES is null, the results will be placed in this global.
1	Connection was closed successfully.
0	Connection failed to be closed.

Table 6-12: API—$$CLOSE^XWBM2MC output

Details:

In addition to the function returning a 1 or 0 indicating success or failure to close the connection to the VistA M server, a 1 or 0 is written to the ^TMP global shown below:

^TMP("XWBM2M",$J,"CONNECTED") = 0

The value written to the ^TMP global can be used as an internal reference for the application.

Example:

SET CLOSE=$$CLOSE^XWBM2MC

If successful:

CLOSE=1

If *not* successful:

CLOSE=0

$$GETCONTX^XWBM2MC—Returns CURRENT Application Context

This API returns the current application context so that a new context may be established, thereby restoring the previous application context prior to switching to the new one. It is an extrinsic function that returns a success/fail indicator of 1 or 0, respectively.

Format:

```
$$GETCONTX^XWBM2MC(.CONTEXT)
```

Input/Output:

Input	Description
CONTEXT	(Required) Variable passed by reference that contains the application context.

Table 6-13: API—$$GETCONTX^XWBM2MC input parameter

Output	Description
1	Current application context was successfully returned.
0	Current application context failed.

Table 6-14: API—$$GETCONTX^XWBM2MC output

Example:

```
SET CCONTEXT=$$GETCONTX^XWBM2MC(.CONTEXT)
```

If successful:

```
    CCONTEXT=1
    CONTEXT=XWB BROKER EXAMPLE
```

If *not* successful:

```
    CCONTEXT=0
```

Chapter 7: Technical Information

This documentation is intended for use in conjunction with the VistA M-to-M Broker, Patch XWB*1.1*34. This is the Technical Manual Section. It details the implementation and maintenance of the M-to-M Broker, as well as routines, options, external and internal relations and software product security for the software.

Implementation and Maintenance

M-to-M Broker is a Kernel Installation and Distribution System (KIDS) software release. M-to-M Broker Installation Instructions can be found in the description for Patch XWB*1.1*34, located on the Patch Module (i.e., Patch User Menu [A1AE USER]) on FOURM.

Software Dependencies

M-to-M Broker requires that both development Test and Production accounts exist in a standard VistA operating environment in order to function correctly. The account(s) must contain the *fully* patched versions of the following software:

- Kernel V. 8.0
- Kernel Toolkit V. 7.3
- The VistA Extensible Markup Language (XML) Parser, Patch XT*7.3*58
- RPC Broker V. 1.1
- VA FileMan V. 22.0

For information on setting up and starting the TCP/IP Service , see the *TCP/IP Supplement, Patch XWB*1.1*35* on the VistA Documentation Library (VDL) at:

http://www.va.gov/vdl/Infrastructure.asp?appID=23

In addition to a standard VistA operating environment, the following patch must be installed before running this patch:

VistA Software and Version	Associated Patch Designation(s)	Brief Patch Description
RPC Broker V. 1.1	XWB*1.1*35	NON-callback server.

Remote Procedure Calls (RPC)

Two Remote Procedure Calls (RPC) used as examples are exported with the M-to-M Broker. They are listed below followed by an explanation of their use.

- XWB M2M EXAMPLE LARRY
- XWB M2M EXAMPLE REF

XWB M2M EXAMPLE LARRY is an sample RPC using all of the M-to-M Broker APIs to illustrate how to build an RPC that will create, accept and return an array. The RPC receives the message, formats the information, and echoes the message back.

XWB M2M EXAMPLE REF is an sample RPC using all of the M-to-M Broker APIs to return a variable by reference.

Routines

This section lists the routines that are exported with M-to-M Broker. All routines are new.

XWBM2MC	XWBRM	XWBUTL
XWBM2MS	XWBRMX	XWBVL
XWBM2MT	XWBRPC	XWBVLC
XWBRL	XWBRPCC	XWBVLL

Options

The option exported with Patch XWB*1.1*34 is named Start M2M RPC Broker Cache Listener [XWB M2M CACHE LISTENER]. It needs to be scheduled in order to start the M2M Broker Listener for Caché. This option is interactive in that the user is prompted to enter the port number. It is recommended that you use port 4800 in the main Production account, which has been reserved for the M-to-M Broker. This option uses the entry point STRT(PORT) to start the M2M Broker Listener.

It is encouraged that VMS/Caché sites use the TCP/IP service.

Archiving and Purging

There are no package-specific archiving or purging procedures or recommendations for the M-to-M Broker.

Callable Routines

This section lists all the APIs exported with the M-to-M Broker. All callable entry points are described in detail in the section titled "VistA M-to-M Broker APIs" in this documentation.

Alphabetized by Entry Point

Entry Point	Brief Description
$$CALLRPC^XWBM2MC	This API builds the Remote Procedure Call (RPC) data structure, and then makes the call to the RPC on the server. The request message is transported in XML and is parsed by using the VistA Extensible Markup Language (XML) Parser, introduced in Kernel Toolkit Patch XT*7.3*58. This API is an extrinsic function returning a success/fail indicator of 1 or 0, respectively.
$$CLOSE^XWBM2MC	This API closes the connection between that particular instance of the "requesting" VistA M server and the "receiving" VistA M server, and does any necessary cleanup. It is an extrinsic function returning a success/fail indicator of 1 or 0, respectively.
$$CONNECT^XWBM2MC	This API establishes the initial connection to the VistA M server. It is an extrinsic function returning a success/fail indicator of 1 or 0, respectively.
$$GETCONTX^XWBM2MC	This API returns the current application context so that a new context may be established, thereby restoring the previous application context prior to switching to the new one. It is an extrinsic function call returning a success/fail indicator of 1 or 0, respectively.
$$GETDIV^XWBM2MC	This API obtains a list of valid divisions for a particular user or logon session. The IEN, station name, and station number are returned for each valid division. If a user has only 1 division, then (XWBDIVG(1)=0) Kernel automatically assigns that division as the default. Use IEN to set division in $$SETDIV.
$$PARAM^XWBM2MC	This API sets up the PARAM data structure necessary to run the RPCs. It is an extrinsic function call returning a success/fail indicator of 1 or 0, respectively.
$$SETCONTX^XWBM2MC	This API sets the context. It sets up the necessary environment to run the RPCs. It is an extrinsic function call returning a success/fail indicator of 1 or 0, respectively.

Entry Point	Brief Description
$$SETDIV^XWBM2MC	This API sets the active division for a particular user or logon session. If only one division is associated with a logon session, (then XWBDIVG(1)=0) Kernel automatically assigns that division as a default.

Table 7-1: Callable entry points exported with the M-to-M Broker

External Interfaces

There are no External Interfaces exported with the M-to-M Broker.

External Relations

Package Requirements

M-to-M Broker requires a standard VistA operating environment in order to function correctly. Check your VistA environment for packages and versions installed.

For more information on the minimum VistA packages and patches that are required by this patch, please refer to the "Software Dependencies" section of this documentation.

Internal Relations

The option exported with Patch XWB*1.1*34 is named Start M2M RPC Broker Cache Listener [XWB M2M CACHE LISTENER]. It needs to be scheduled in order to start the M2M Broker Listener for Caché. This option is interactive in that the user is prompted to enter the port number. It is recommended that you use port 4800 in the main Production account, which has been reserved for the M-to-M Broker. This option uses the entry point STRT(PORT) to start the M2M Broker Listener.

It is encouraged that VMS/Caché sites use the TCP/IP service.

Namespace

M-to-M Broker has been assigned the **XWB** namespace, which is shared with the RPC Broker namespace.

Software Product Security

Mail Groups

There are no mail groups exported with M-to-M Broker.

Remote Systems

Connections

The M-to-M Broker transmits data using TCP/IP, allowing connections from other VistA M servers. Connection by those VistA M servers is subject to authentication as any normal logon requires. VistA applications can use any remote procedure call (RPC) authorized to the application, if the application is authorized to the signed-on user. Data is exchanged between VistA M servers, which can be anywhere on VA's TCP/IP network. This data is bundled in XML and parsed out using the VistA Extensible Markup Language (XML) Parser.

Encryption is used when a user's Access and Verify codes are sent between VistA M servers.

For information on VistA Extensible Markup Language (XML) Parser, Kernel Toolkit Patch XT*7.3*58, please refer to the "VistA Extensible Markup Language (XML) Parser Technical and User Documentation", located at: http://vista.med.va.gov/vdl/Infrastructure.asp#App12 .

Archiving/Purging

There are no package-specific archiving or purging procedures or recommendations for the M-to-M Broker.

Interfacing

No *non*-VA products are embedded in or required by the M-to-M Broker, other than those provided by the underlying operating systems.

Electronic Signatures

Electronic signatures are not used within the M-to-M Broker.

Security Keys

No security keys are exported with M-to-M Broker.

Glossary

ACCESS CODE	A code that, along with the Verify code, allows the computer to identify you as a user authorized to gain access to the computer. Your code is greater than 6 and less than 20 characters long; can be numeric, alphabetic, or a combination of both; and is usually assigned by a site manager or application coordinator. It is used by the Kernel's Sign-on/Security system to identify the user (see Verify Code).
ALERTS	Brief online notices that are issued to users as they complete a cycle through the menu system. Alerts are designed to provide interactive notification of pending computing activities, such as the need to reorder supplies or review a patient's clinical test results. Along with the alert message is an indication that the View Alerts common option should be chosen to take further action.
ANSI MUMPS	The MUMPS programming language is a standard recognized by the **A**merican **N**ational **S**tandard **I**nstitute (ANSI). MUMPS stands for **M**assachusetts **U**tility **M**ulti-**p**rogramming **S**ystem and is abbreviated as M.
API	Application Programmer Interface. VistA Application Programmer Interfaces (APIs) are units of programming code provided by a custodial development domain to permit developers outside the custodial domain to accomplish a specified purpose. In some programming languages, APIs are called (sub)routines. APIs in VistA may be defined as extrinsic functions, extrinsic special variables, or label references to routines. VistA APIs fall into the following three categories: 1. The first category is "Supported API" These are callable routines, which are supported for general use by all VistA applications. 2. The second category is "Controlled Subscription API." These are callable routines for which you must obtain an Integration Agreement (IA - formerly referred to as a DBIA) to use. 3. The third category is "Private API," where only a single application is granted permission to use an attribute/function of another VistA package. These IAs are granted for special cases, transitional problems between versions, and release coordination.
APPLICATION PACKAGE	Software and documentation that support the automation of a service, such as Laboratory or Pharmacy within VA medical centers. The Kernel application package is like an operating system relative to other VistA applications.
CALLABLE ENTRY POINT	An authorized programmer call that may be used in any VistA application package. The DBA maintains the list of DBIC-approved entry points.

CARET	A symbol expressed as up caret ("^"), left caret ("<"), or right caret (">"). In many M systems, a right caret is used as a system prompt and an up caret as an exiting tool from an option. Also known as the up-arrow symbol or shift–6 key.
CLIENT	A single term used interchangeably to refer to the user, the workstation, and the portion of the program that runs on the workstation. This term is typically used in an object-oriented environment, where a client is a member of a group that uses the services of an unrelated group. If the client is on a local area network (LAN), it can share resources with another computer (server). With respect to the M-to-M Broker software, client refers to the "requesting server" that is able to connect to a "receiving server," where both servers reside in VistA on the same or on different VistA M systems.
COMPONENT	An object-oriented term used to describe the building blocks of GUI applications. A software object that contains data and code. A component may or may not be visible. These components interact with other components on a form to create the GUI user application interface.
CONTROLLED SUBSCRIPTION INTEGRATION AGREEMENT	This applies where the IA describes attributes/functions that must be controlled in their use. The decision to restrict the IA is based on the maturity of the custodian package. Typically, these IAs are created by the requesting package based on their independent examination of the custodian package's features. For the IA to be approved, the custodian grants permission to other VistA packages to use the attributes/functions of the IA; permission is granted on a one-by-one basis where each is based on a solicitation by the requesting package. An example is the extension of permission to allow a package (e.g., Spinal Cord Dysfunction) to define and update a component that is supported within the Health Summary package file structures.
COTS	**C**ommercial **O**ff-**t**he-**S**helf. COTS refers to software packages that can be purchased by the public and used in support of VistA.
DATA DICTIONARY	The Data Dictionary is a global containing a description of the kind of data that is stored in the global corresponding to a particular file. VA FileMan uses the data internally for interpreting and processing files. A Data Dictionary (DD) contains the definitions of a file's elements (fields or data attributes), relationships to other files, and structure or design. Users generally review the definitions of a file's elements or data attributes; programmers review the definitions of a file's internal structure.
DBIA	**D**ata**b**ase **I**ntegration **A**greement, a formal understanding between two or more application packages that describes how data is shared or how packages interact. The DBA maintains a list of DBIAs between package developers, allowing the use of internal entry points or other package-specific features that are not available to the general programming public.
DDP	**D**istributed **D**ata **P**rocessing

DEFAULT	A response the computer considers the most probable answer to the prompt being given. In the roll-and-scroll mode of VistA, the default value is identified by double forward slash marks (//) immediately following it. In a GUI-based application the default may be a highlighted button or text. This allows you the option of accepting the default answer or entering your own answer. To accept the default you simply press the enter (or return) key. To change the default answer, type in your response.
DICOM	Digital Imaging and Communication in Medicine
DIRECT MODE UTILITY	A programmer call that is made when working in direct programmer mode. A direct mode utility is entered at the M prompt (e.g., >**D ^XUP**). Calls that are documented as direct mode utilities *cannot* be used in application package code.
DLL	**D**ynamic **L**ink **L**ibrary. A DLL allows executable routines to be stored separately as files with a DLL extension. These routines are only loaded when a program calls for them. DLLs provide several advantages: 1. DLLs help save on computer memory, since memory is only consumed when a DLL is loaded. They also save disk space. With static libraries, your application absorbs all the library code into your application so the size of your application is greater. Other applications using the same library will also carry this code around. With the DLL, you don't carry the code itself, you have a pointer to the common library. All applications using it will then share one image. 2. DLLs ease maintenance tasks. Because the DLL is a separate file, any modifications made to the DLL will not affect the operation of the calling program or any other DLL. 3. DLLs help avoid redundant routines. They provide generic functions that can be utilized by a variety of programs.
ERROR TRAP	A mechanism to capture system errors and record facts about the computing context such as the local symbol table, last global reference, and routine in use. Operating systems provide tools such as the %ER utility. The Kernel provides a generic error trapping mechanism with use of the ^%ZTER global and ^XTER* routines. Errors can be trapped and, when possible, the user is returned to the menu system.
FORUM	The central e-mail system within VistA. Developers use FORUM to communicate at a national level about programming and other issues. FORUM is located at the Washington, DC CIO Field Office (162-2).
GUI	**G**raphical **U**ser **I**nterface. A type of display format that enables users to choose commands, initiate programs, and other options by selecting pictorial representations (icons) via a mouse or a keyboard.
HIS	**H**ospital **I**nformation **S**ystem

HOST	The term Host is used interchangeably with the term Server.
ICON	A picture or symbol that graphically represents an object or a concept.
INTEGRATION AGREEMENTS (IA) (Formerly known as DATABASE INTEGRATION AGREEMENTS [DBIA])	Integration Agreements define an agreement between two or more VistA packages to allow access to one development domain by another. Any package developed for use in the VistA environment is required to adhere to this standard; as such it applies to vendor products developed within the boundaries of DBA assigned development domains (e.g., MUMPS AudioFax). An IA defines the attributes and functions that specify access. All IAs are recorded in the Integration Agreement database on FORUM. Content can be viewed using the DBA menu or the Technical Services' Web page.
IRM	**I**nformation **R**esource **M**anagement. A service at VA medical centers responsible for computer management and system security.
KERNEL	A set of VistA software routines that function as an intermediary between the host operating system and the VistA application packages (e.g., Laboratory, Pharmacy, IFCAP, etc.). Kernel provides a standard and consistent user and programmer interface between application packages and the underlying M implementation. (VA FileMan and MailMan are self-contained to the extent that they can standalone as verified packages.) Some of Kernel's components are listed below along with their associated namespace assignments:

KIDS	XPD
Menu Management	XQ
Tools	XT
Sign-on/Security	XU
Device Handling	ZIS
Task Management	ZTM

LISTENER	M-to-M Broker does not use the original Broker Listener. Instead, M-to-M Broker introduces a new listener for VMS/Caché operating systems, which is being provided by a Transmission Control Protocol/Internet Protocol (TCP/IP) service. This service will establish a connection using TCP/IP on a VMS/Caché system.

For sites running on VMS/Caché, in order to utilize M-to-M communications, it's necessary to start this TCP/IP Service.

MENU MANAGER	The Kernel module that controls the presentation of user activities such as menu choices or options. Information about each user's menu choices is stored in the Compiled Menu System, the ^XUTL global, for easy and efficient access.
MSM	Micronetics Standard MUMPS

MULTIPLE	A multiple-valued field; a subfile. In many respects, a multiple is structured like a file.
MUMPS (ANSI STANDARD)	A programming language recognized by the **A**merican **N**ational **S**tandards **I**nstitute (ANSI). The acronym MUMPS stands for **M**assachusetts General Hospital **U**tility **M**ulti-**p**rogramming **S**ystem and is abbreviated as M.
NAMESPACING	A convention for naming VistA package elements. The Database Administrator (DBA) assigns unique character strings for package developers to use in naming routines, options, and other package elements so that packages may coexist. The DBA also assigns a separate range of file numbers to each package.
NODE	In a tree structure, a point at which subordinate items of data originate. An M array element is characterized by a name and a unique subscript. Thus the terms: node, array element, and subscripted variable are synonymous. In a global array, each node might have specific fields or "pieces" reserved for data attributes such as name.
NT	New Technology
OIFO	Office of Information Field Office
OPTION	As an item on a menu, an option provides an opportunity for users to select it, thereby invoking the associated computing activity. In VistA, an entry in the OPTION file (#19). Options may also be scheduled to run in the background, non-interactively, by TaskMan.
PRIVATE INTEGRATION AGREEMENT	Where only a single application is granted permission to use an attribute/function of another VistA package. These IAs are granted for special cases, transitional problems between versions, and release coordination. A Private IA is also created by the requesting package based on their examination of the custodian package’s features. An example would be where one package distributes a patch from another package to ensure smooth installation.
PROMPT	The computer interacts with the user by issuing questions called *prompts*, to which the user returns a response.
REMOTE PROCEDURE CALL (RPC)	A remote procedure call (RPC) is essentially M code that may take optional parameters to do some work and then return either a single value or an array back to the client application.
ROUTINE	A program or a sequence of instructions called by a program that may have some general or frequent use. M routines are groups of program lines that are saved, loaded, and called as a single unit via a specific name.

SECURITY KEY	The purpose of Security Keys is to set a layer of protection on the range of computing capabilities available with a particular software package. The availability of options is based on the level of system access granted to each user.
SERVER	With respect to the M-to-M Broker software, server refers to the "receiving server" that sends the results in a message back to the "requesting server," where both servers reside in VistA on the same or on different VistA M systems. The server is where VistA M-based data and Business Rules reside, making these resources available to the requesting server. When the requesting server is receiving the results, it is referred to as the "server."
SIGN-ON/SECURITY	The Kernel module that regulates access to the menu system. It performs a number of checks to determine whether access can be permitted at a particular time. A log of signons is maintained.
SUBSCRIPT	A symbol that is associated with the name of a set to identify a particular subset or element. In M, a numeric or string value that: is enclosed in parentheses, is appended to the name of a local or global variable, and identifies a specific node within an array.
SUPPORTED REFERENCE INTEGRATION AGREEMENT	This applies where any VistA application may use the attributes/functions defined by the IA (these are also called "**Public** "). An example is an IA that describes a standard API such as DIE or VADPT. The package that creates/maintains the Supported Reference must ensure it is recorded as a Supported Reference in the IA database. There is no need for other VistA packages to request an IA to use these references; they are open to all by default.
TCP/IP	Transmission Control Protocol/Internet Protocol
UCI	**U**ser **C**lass **I**dentification, a computing area. The MGR UCI is typically the Manager's account, while VAH or ROU may be Production accounts.
USER ACCESS	This term is used to refer to a limited level of access to a computer system that is sufficient for using/operating a package, but does not allow programming, modification to data dictionaries, or other operations that require programmer access. Any of VistA's options can be locked with a security key (e.g., XUPROGMODE, which means that invoking that option requires programmer access). The user's access level determines the degree of computer use and the types of computer programs available. The Systems Manager assigns the user an access level.

USER INTERFACE	The way the package is presented to the user, such as Graphical User Interfaces that display option prompts, help messages, and menu choices. A standard user interface can be achieved by using Borland's Delphi Graphical User Interface to display the various menu option choices, commands, etc.
VA	Veterans Administration
VERIFY CODE	The Kernel's Sign-on/Security system uses the Verify code to validate the user's identity. This is an additional security precaution used in conjunction with the Access code. Verify codes shall be at least eight characters in length and contain three of the following four kinds of characters: letters (lower- and uppercase), numbers, and, characters that are neither letters nor numbers (e.g., "**#**", "**@**" or "**$**"). If entered incorrectly, the system does not allow the user to access the computer. To protect the user, both codes are invisible on the terminal screen.
VHA	Veterans Health Administration
VISN	Veterans Integrated Service Network
VistA	**V**eterans Health **I**nformation **S**ystems and **T**echnology **A**rchitecture. VistA includes the VA's application software (i.e., Microsoft Windows-based and locally-developed applications, roll-and-scroll, and interfaces such as software links to commercial packages). In addition, it encompasses the VA's uses of new automated technology including the clinical workstations. VistA encompasses the rich automated environment already present at local VA medical facilities.
WINDOW	An object on the screen (dialog) that presents information such as a document or message.
XML	Extensible Markup Language. The universal format for structured documents and data on the Web.

Appendix A: Error Messages

This section describes the error messages associated with M-to-M Broker APIs. Error messages encountered during M-to-M Broker Client/Server processing within the VistA environment are recorded in the ^TMP global in API specific subscripts. This section documents these error messages, listed alphabetically including the global location, associated API, and a brief description.

Error Msg.	**Control Character Found**
Global	^TMP(“XWBM2ME”,$J,”ERROR”,”CALLRPC”)
API	$$CALLRPC^XWBM2MC—Build the Remote Procedure Data Structure
Description	Failed attempt at building and/or making the call to the RPC on the server, or transporting the request message in XML and/or parsing it using the VistA Extensible Markup Language (XML) Parser.

Table A-1: Error message—Control Character Found

Error Msg.	**Could not obtain list of valid divisions for current user**
Global	^TMP("XWBM2ME",$J,"ERROR",”GETDIV”)
API	$$GETDIV^XWBM2MC—Get Division for Logon Session
Description	Failed attempt to obtain a list of valid divisions for the current user or logon session.

Table A-2: Error message—Could not obtain list of valid divisions for current user

Error Msg.	**Could not open connection**
Global	^TMP("XWBM2ME",$J,"ERROR",”CONNECT”)
API	$$CONNECT^XWBM2MC—M Client/Server Connection
Description	Failed attempt to establish the connection to the VistA M Server.

Table A-3: Error message—Could not open connection

Error Msg.	**Could not Set active Division for current user**
Global	^TMP("XWBM2ME",$J,"ERROR",”SETDIV”)
API	$$SETDIV^XWBM2MC—Set Division for Logon Session
Description	Failed attempt to set the active Division for the current user or logon session.

Table A-4: Error message—Could not Set active Division for current user

Error Msg.	**Invalid user, no DUZ returned**
Global	^TMP("XWBM2ME",$J,"ERROR",”SIGNON”)
API	$$CONNECT^XWBM2MC—M Client/Server Connection
Description	User was not authenticated during logon. No DUZ was returned.

Table A-5: Error message—Invalid user, no DUZ returned

Error Msg.	**RPC could not be processed**
Global	^TMP(“XWBM2ME”,$J,”ERROR”,”CALLRPC”)
API	$$CALLRPC^XWBM2MC—Build the Remote Procedure Data Structure
Description	Failed attempts at building and/or making the call to the RPC on the server, or transporting the request message in XML and parsing it using the VistA Extensible Markup Language (XML) Parser. Example: The RPC may not have been found because the user entered it incorrectly.

Table A-6: Error message—RPC could not be processed

Error Msg.	**There is no connection**
Global	^TMP(“XWBM2ME”,$J,”ERROR”,”CALLRPC”)
API	$$CALLRPC^XWBM2MC—Build the Remote Procedure Data Structure
Description	Failed attempts at building and/or making the call to the RPC on the server, or transporting the request message in XML and parsing it using the VistA Extensible Markup Language (XML) Parser. Example: There was no physical connection. Therefore, no RPC could be run.

Table A-7: Error message—There is no connection

Error Msg.	**XUS AV CODE RPC failed**
Global	^TMP("XWBM2ME",$J,"ERROR",”SIGNON”)
API	$$CONNECT^XWBM2MC—M Client/Server Connection
Description	Failed attempt to establish the connection to the VistA M Server. Example: because Access and Verify codes were incorrect.

Table A-8: Error message—XUS AV CODE RPC failed

Error Msg.	**XUS SIGNON SETUP RPC failed**
Global	^TMP("XWBM2ME",$J,"ERROR",”SIGNON”)
API	$$CONNECT^XWBM2MC—M Client/Server Connection
Description	VistA Access and Verify codes failed. Could not set up the appropriate environment for that logon session. This is a Kernel RPC.

Table A-9: Error message—XUS SIGNON SETUP RPC failed

Error Msg.	**Remote Procedure Unknown**
Global	^TMP("XWBM2ME",$J,"ERROR",”SERVER”)
API	$$CALLRPC^XWBM2MC—M Client/Server Connection
Description	RPC could not be found. Make sure the RPC is spelled correctly. The RPC name must be correctly entered in the input parameter RPCNAM, which is passed into the $$CALLRPC^XWBM2MC API.

Table A-10: Error message—Remote Procedure Unknown

Index

A

Anonymous Directories, xi
APIs
  Callable Routines, 7-3
  definition, 1
  M-to-M Broker, 6-1
application context, 5-1
  $$GETCONTX^XWBM2MC, 6-14, 7-3
  $$GETDIV^XWBM2MC, 6-5
  $$SETCONTX^XWBM2MC, 5-2, 6-4, 7-3, 7-4
  $$SETDIV^XWBM2MC, 6-7
  restore original context, 6-14, 7-3
  return current context, 6-14, 7-3
  set context, 6-4, 6-5, 6-7, 7-3, 7-4
  Switching Between Application Contexts, 5-3
  When is it Not Necessary to Set the Application Context?, 5-3
application entry points, 7-3
Application Programmer Interfaces (API)
  $$$$GETDIV^XWBM2MC, 5-3
  $$CALLRPC^XWBM2MC, 5-4, 6-11, 7-3
  $$CLOSE^XWBM2MC, 6-13, 7-3
  $$CONNECT^XWBM2MC, 6-2, 7-3
  $$GETCONTX^XWBM2MC, 6-14, 7-3
  $$GETDIV^XWBM2MC, 6-5
  $$PARAM^XWBM2MC, 6-9, 7-3
  $$SETCONTX^XWBM2MC, 6-4, 6-7, 7-3, 7-4
  $$SETDIV^XWBM2MC, 6-7
Approved Application Abbreviations, ix
Archiving and Purging, 7-3
Assumptions About the Reader, x

B

B-type option, 4-1, 4-2, 5-2
Build and Request an RPC to Run
  $$CALLRPC^XWBM2MC, 5-3
  $$PARAM^XWBM2MC, 5-3

C

Callable entry points, 7-3
Callable Routines, 7-3
$$CALLRPC^XWBM2MC, 5-3, 5-4, 6-11, 7-3
Client, definition, 2
Client/Server functionality, 6-1
Close the VistA Server Connection
  $$CLOSE^XWBM2MC, 5-4
$$CLOSE^XWBM2MC, 5-4, 6-13, 7-3
Component, definition, 2
$$CONNECT^XWBM2MC, 5-1, 6-2, 7-3
connection to server
  close, 5-1
  $$CLOSE^XWBM2MC, 6-13, 7-3
  $$CONNECT^XWBM2MC, 6-2, 7-3
  Access and Verify codes, 6-2, 7-3
  close connection, 6-13, 7-3
  Close the VistA Server Connection, 5-4
  establish connection, 6-2, 7-3, 5-1
  IP address, 6-2, 7-3
  PORT, 6-2, 7-3
  TCP/IP
    Service, 6-2
  TCP/IP Service, 7-3
  When do I Leave the Connection Open?, 5-5
Contents, Table of, v
context, application, 5-2
  B-type option, 4-1, 5-2
  Switching Between Application Contexts, 5-3
  When is it Not Necessary to Set the Application Context?, 5-3
Control Character Found
  failed call to RPC on server, 1
Control Character Handling, 5-5
Could not obtain list of valid divisions for current user
  cannot obtain divisions, 1
Could not open connection
  cannot connect to VistA, 1
create your own RPCs, 3-1

D

data structure
  $$PARAM^XWBM2MC, 6-9, 7-3
  set up data structure, 6-9, 7-3
DDP
  definition, 2
  protocol, xiv
DICOM, xiii
  definition, 3
Documentation History, iii
Documentation Symbols, ix

E

Encryption, 7-5
Entry points, 7-3
Error Messages
  Control Character Found, 1
  Could not obtain list of valid divisions for current user, 1
  Could not open connection, 1

Could not Set active Division for current user, 1
Invalid user, no DUZ returned, 2
Remote Procedure Unknown, 3
RPC could not be processed, 2
There is no connection, 2
XUS AV CODE RPC failed, 2
XUS SIGNON SETUP RPC failed, 3
Establish the Connection to the VistA M Sever
$$CONNECT^XWBM2MC, 5-1
EVS Anonymous Directories, xi
Extensible Markup Language (XML), 3-1
External Relations, 7-4

F

Figures, Table of, vii
files
OPTION (#19), 4-1, 5-2
REMOTE PROCEDURE (#8994), 3-1
FTP directories, xi

G

$$GETCONTX^XWBM2MC, 5-3, 6-14, 7-3
$$GETDIV^XWBM2MC, 5-3, 6-5

H

Home Pages
HSD&D Home Page Web address, x
M-to-M Broker Web address, xi
Host, definition, 4
How to Run an M-to-M Broker RPC, 5-1, 4-2–5-5
Build and Request an RPC to Run, 5-3
Close the VistA Server Connection, 5-4
Set Up the Environment to Run the RPCs in VistA, 5-2
Switching Between Application Contexts, 5-3
Using $$PARAM^XWBM2MC With $$CALLRPC^XWBM2MC, 5-4
Using Standalone $$CALLRPC^XWBM2MC, 5-4
What is a VistA Application Context?, 5-2
When do I Leave the Connection Open?, 5-5
When is it Not Necessary to Set the Application Context?, 5-3

I

Imaging Service, xiii
Implementation and Maintenance, 7-1
input parameters
$$CALLRPC^XWBM2MC, 6-11
$$CONNECT^XWBM2MC, 6-2
$$GETCONTX^XWBM2MC, 6-14
$$PARAM^XWBM2MC, 6-9
$$SETCONTX^XWBM2MC, 6-4
$$SETDIV^XWBM2MC, 6-7
Internal Relations, 7-4
Introduction, xiii
Invalid user, no DUZ returned
user not authenticated, 2

K

Kernel Toolkit Patch XT*7.3*58, 5-4, 6-11, 7-3

L

Listener
TCP/IP Service, 4
Listener, definition, 4
Listeners for VMS/Caché Operating Systems, 2-1

M

Message Structure, XML, 3-1
M-to-M Broker, xiii
APIs, 6-1
new implementation of the RPC Broker, xiii
M-to-M Broker RPC, How to Run an, 5-1
Build and Request an RPC to Run, 5-3
Close the VistA Server Connection, 5-4
Establish the Connection to the VistA M Sever, 5-1
Set Up the Environment to Run the RPCs in VistA, 5-2
Switching Between Application Contexts, 5-3
Using $$PARAM^XWBM2MC With $$CALLRPC^XWBM2MC, 5-4
Using Standalone $$CALLRPC^XWBM2MC, 5-4
What is a VistA Application Context?, 5-2
When do I Leave the Connection Open?, 5-5
When is it Not Necessary to Set the Application Context?, 5-3

N

Namespace, 7-4
New Listener, 2-1

O

Obtain and Set the Division for the Current User or Logon Session
$$GETDIV^XWBM2MC, 5-3
OPTION file (#19), 4-1, 5-2
options
B-type option, 4-1, 4-2, 5-2
Options, 7-2
Orientation, ix
conventions for displaying TEST data, ix
EVS Anonymous Directories, xi

output
  $$CALLRPC^XWBM2MC, 6-11
  $$CLOSE^XWBM2MC, 6-13
  $$CONNECT^XWBM2MC, 6-2
  $$GETCONTX^XWBM2MC, 6-14
  $$GETDIV^XWBM2MC, 6-5
  $$PARAM^XWBM2MC, 6-9
  $$SETCONTX^XWBM2MC, 6-4
  $$SETDIV^XWBM2MC, 6-7

P

$$PARAM^XWBM2MC, 5-3, 5-4, 6-9, 7-3
Patch History, iii
Patch XT*7.3*58, 5-4, 6-11, 7-3
Patch XWB*1.1*34, xi
patient & user names
  test data, ix

R

Reader, Assumptions About the, x
Remote Procedure Call (RPC)
  call to RPC on server, 5-3, 5-4, 6-11, 7-3
  create your own RPCs, 3-1
  definition, 5
  failed call to RPC on server, 1, 2
  Validating RPCs, 4-1
Remote Procedure Calls (RPC), 7-2
REMOTE PROCEDURE file (#8994), 3-1
Remote Procedure Unknown
  RPC incorrectly spelled, 3
  RPC not found, 3
  RPCNAM, 3
Remote Systems, 7-5
Revision History
  Documentation, iii
Routines, 7-2
RPC Broker, xiv
RPC could not be processed
  failed call to RPC on server, 2
RPC is registered to an application, 4-2
RPC Registration, 4-2
RPC, How to Run an M-to-M Broker
  Build and Request an RPC to Run, 5-3
  Close the VistA Server Connection, 5-4
  Establish the Connection to the VistA M Sever, 5-1
  Set Up the Environment to Run the RPCs in VistA, 5-2
  Switching Between Application Contexts, 5-3
  Using $$PARAM^XWBM2MC With $$CALLRPC^XWBM2MC, 5-4
  Using Standalone $$CALLRPC^XWBM2MC, 5-4
  What is a VistA Application Context?, 5-2
  When do I Leave the Connection Open?, 5-5
  When is it Not Necessary to Set the Application Context?, 5-3

S

Security, 4-1–4-2
  Sample Security Procedures, 4-2
  Summary of Tasks, 4-2
  Validating RPCs, 4-1
server connection
  $$CLOSE^XWBM2MC, 6-13, 7-3
  $$CONNECT^XWBM2MC, 6-2, 7-3
  Access and Verify codes, 6-2, 7-3
  close connection, 6-13, 7-3
  establish connection, 6-2, 7-3
  IP address, 6-2, 7-3
  PORT, 6-2, 7-3
  TCP/IP Service, 6-2, 7-3
Server, definition, 6
Set Up the Environment to Run the RPCs in VistA
  $$SETCONTX^XWBM2MC, 5-2
$$SETCONTX^XWBM2MC, 5-2, 6-4, 7-3, 7-4
$$SETDIV^XWBM2MC, 6-7
Social Security Numbers
  test data, ix
Software Dependencies, 1-1, 7-1
Software Product Security
  Remote Systems, 7-5
Supported References, 6-1
Switching Between Application Contexts
  $$GETCONTX^XWBM2MC, 5-3
Symbols Found in the Documentation, ix
System Features
  Message Structure, XML, 3-1
  Security, 4-1–4-2
  TCP/IP Service, 2-1

T

Table of Contents, v
Table of Figures, vii
TCP/IP
  create service, 2-1
  definition, 6
  Service, x
Technical Information
  Archiving and Purging, 7-3
  Callable Routines, 7-3
  External Relations, 7-4
  Implementation and Maintenance, 7-1
  Internal Relations, 7-4
  Namespace, 7-4
  Options, 7-2
  Remote Procedure Calls (RPC), 7-2
  Routines, 7-2
  Software Dependencies, 7-1

Software Product Security, 7-5
test data
patient & user names, ix
Social Security Numbers, ix
There is no connection
failed call to RPC on server, 2
no physical connection established, 2

U

URLs
Getting Started With The Broker Development Kit (BDK), 3-1, 5-1
HSD&D Home Page Web address, x
M-to-M Broker documentation, xi
M-to-M Broker Web address, xi
RPC Broker documentation, xiv
VistA XML Parser documentation, 3-1, 7-5
Use Case
Build and Request an RPC to Run, 5-3
Close the VistA Server Connection, 5-4
Control Character Handling, 5-5
Establish the Connection to the VistA M Sever, 5-1
How to Run an M-to-M Broker RPC, 5-1
Obtain and Set the Division for the Current User or Logon Session, 5-3
Set Up the Environment to Run the RPCs in VistA, 5-2
Using $$PARAM^XWBM2MC With $$CALLRPC^XWBM2MC, 5-4
Using Standalone $$CALLRPC^XWBM2MC, 5-4

V

valid connection request, 4-2
valid user, 4-2
Validating
RPCs, Security, 4-1
VistA Imaging Service, xiii
VistA M-to-M Broker, xiii
APIs, 6-1
VistA XML Parser documentation, 3-1, 7-5

W

Web pages
Getting Started With The Broker Development Kit (BDK), 3-1, 5-1
HSD&D Home Page Web address, x
M-to-M Broker documentation, xi
M-to-M Broker Web address, xi
RPC Broker documentation, xiv
VistA XML Parser documentation, 3-1, 7-5
What is a VistA Application Context?
$$SETCONTX^XWBM2MC, 5-2
When do I Leave the Connection Open?, 5-5
When is it Not Necessary to Set the Application Context?, 5-3

X

XML message structure, 3-1
call to RPC on server, 5-3, 5-4, 6-11, 7-3
failed call to RPC on server, 1, 2
XML, definition, 7
XUS AV CODE RPC failed
cannot connect to VistA, 2
incorrect Access/Verify codes, 2
XUS SIGNON SETUP RPC failed
Access/Verify codes failed, 3
XWB*1.1*34, x, xi