RPC Broker

User Guide

Software Version 1.1

September 1997

Revised April 2014

Department of Veterans Affairs (VA)

Office of Information and Technology (OIT)

Product Development (PD)

Revision History

Document Revisions

Table 1. Documentation revision history

Date	Revision	Description	Author
04/10/2014	5.3	Tech Edit: • Made minor format and style corrections throughout. • Updated input parameter types (PType values) in Table 7. • Made minor reference updates.	• Technical Writer: T. B.
03/26/2014	5.2	Tech Edit: • Changed references from "Broker.hlp" to "Broker_1_1.chm" throughout. • Updated other help references and instructions related to updated Broker help; replacing WinHelp (Broker.hlp) to HTML help (Broker_1_1.zip with Broker_1_1.chm file and Broker_1_1-HTML_Files.zip with multiple HTML files). • Changed references from "programmer" to "developer" throughout. • Made other minor grammar and punctuation corrections throughout.	• Technical Writer: T. B.
12/04/2013	5.1	Tech Edit: • Updated document for RPC Broker Patch XWB*1.1*50 based on feedback from H. W. • Removed references related to Virgin Installations throughout. • Updated file name references throughout. • Removed distribution files that are obsolete or no longer distributed throughout. • Updated RPC Broker support on the following software: o Microsoft® XP and 7.0 (operating system) throughout. o Microsoft® Office Products 2010 throughout. o Changed references from "Borland"	• Developer: H. W. • Technical Writer: T. B.

Date	Revision	Description	Author
		to "Embarcadero" and updated support for Delphi Versions XE5, XE4, XE3, and XE2 throughout. • Updated all images for prior Microsoft® Windows operating systems to Windows 7 dialogues. • Deleted Section 6, "RPC Broker Developer Utilities," since those utilities no longer exist in this latest version of the Broker. • Updated the "RPC Broker and Delphi" section for Delphi XE5, XE4, XE3, and XE2. • Removed sample DLL from Section 7. • Redacted document for the following information: o Names (replaced with role and initials). o Production IP addresses and ports. o Intranet websites. **RPC Broker 1.1**	
07/25/2013	5.0	Tech Edit: • Baselined document. • Updated all styles and formatting to follow current internal team style template. • Updated all organizational references.	• Developer: H. W. • Technical Writer: T. B.
08/26/2008	4.2	Updates for RPC Broker Patch XWB*1.1*50: • Added new properties. • Support for Delphi 5, 6, 7, 2005, 2006, and 2007. • Changed references form Patch 47 to Patch 50 where appropriate.	• Project Manager: J. Sch. • Developer: J. I. • SQA: G. S. • Technical Writer: T. B.
07/03/2008	4.1	Updates for RPC Broker Patch XWB*1.1*47: • No content changes required; no new public classes, methods, or properties added to those available in XWB*1.1*40. • Bug fixes to the ValidAppHandle function and fixed memory leaks. • Support added for Delphi 2005, 2006, and 2007. • Reformatted document. • Changed references form Patch 40 to Patch 47 where appropriate.	• Common Services (CS) Development Team Oakland, CA OIFO: • Project Manager: J. Sch. • Developer: J. I. • SQA: G. S. • Technical Writer: T. B.
02/24/2005	4.0	Revised Version for RPC Broker Patches	• Developer: J. I.

Date	Revision	Description	Author
		XWB*1.1*35 and 40. Also, reviewed document and edited for the "Data Scrubbing" and the "PDF 508 Compliance" projects. **Data Scrubbing—**Changed all patient/user TEST data to conform to standards and conventions as indicated below: • The first three digits (prefix) of any Social Security Numbers (SSN) start with "000" or "666." • Patient or user names are formatted as follows: XWBPATIENT,[N] or XWBUSER,[N] respectively, where the N is a number written out and incremented with each new entry (e.g., XWBPATIENT, ONE, XWBPATIENT, TWO, etc.). • Other personal demographic-related data (e.g., addresses, phones, IP addresses, etc.) were also changed to be generic. **PDF 508 Compliance—**The final PDF document was recreated and now supports the minimum requirements to be 508 compliant (i.e., accessibility tags, language selection, alternate text for all images/icons, fully functional Web links, successfully passed Adobe Acrobat Quick Check).	• Technical Writer: T. B.
05/08/2002	3.0	Revised Version for RPC Broker Patch XWB*1.1*26.	• Developer: J. I. • Technical Writer: T. B.
05/01/2002	2.0	Revised Version for RPC Broker Patch XWB*1.1*13.	• Developer: J. I. • Technical Writer: T. B.
09/--/1997	1.0	Initial RPC Broker Version 1.1 software release.	• Developer: J. I. • Technical Writer: T. B.

Patch Revisions

For the current patch history related to this software, see the Patch Module on FORUM.

Contents

Figures and Tables

Figures

Tables

Orientation

How to Use this Manual

Throughout this manual, advice and instructions are offered regarding the use of the Remote Procedure Call (RPC) Broker 1.1 Development Kit (BDK) and the functionality it provides for Veterans Health Information Systems and Technology Architecture (VistA).

Intended Audience

The intended audience of this manual is the following stakeholders:

- Product Development (PD)—VistA legacy development teams.
- Information Resource Management (IRM)—System administrators at Department of Veterans Affairs (VA) sites who are responsible for computer management and system security on the VistA M Server.
- Information Security Officers (ISOs)—Personnel at VA sites responsible for system security.
- Health Product Support (HPS).

Legal Requirements

There are no special legal requirements involved in the use of the RPC Broker.

Disclaimers

This manual provides an overall explanation of configuring RPC Broker and the functionality contained in RPC Broker 1.1; however, 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 Internet and Intranet SharePoint sites and Websites for a general orientation to VistA. For example, visit the Office of Information and Technology (OIT) Product Development (PD) Intranet Website.

DISCLAIMER: The appearance of any external hyperlink references in this manual does not constitute endorsement by the Department of Veterans Affairs (VA) of this Website 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.

Documentation Conventions

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:

Table 2. Documentation symbol descriptions

Symbol	Description
ℹ	**NOTE/REF:** Used to inform the reader of general information including references to additional reading material.
⚠	**CAUTION / RECOMMENDATION / DISCLAIMER:** Used to caution the reader to take special notice of critical information.

- Descriptive text is presented in a proportional font (as represented by this font).
- Conventions for displaying TEST data in this document are as follows:
  - The first three digits (prefix) of any Social Security Numbers (SSN) begin with either "000" or "666."
  - Patient and user names are formatted as follows: [Application Name]PATIENT,[N] and [Application Name]USER,[N] respectively, where "Application Name" is defined in the Approved Application Abbreviations document and "N" represents the first name as a number spelled out and incremented with each new entry. For example, in RPC Broker (XWB) test patient and user names would be documented as follows: XWBPATIENT,ONE; XWBPATIENT,TWO; XWBPATIENT,THREE; etc.
- "Snapshots" of computer online displays (i.e., screen captures/dialogues) and computer source code are shown in a *non*-proportional font and may be enclosed within a box.
- User's responses to online prompts are **bold** typeface and highlighted in yellow (e.g., **<Enter>**).
- Emphasis within a dialogue box is **bold** typeface and highlighted in blue (e.g., STANDARD LISTENER: RUNNING).
- Some software code reserved/key words are **bold** typeface with alternate color font.
- References to "<**Enter**>" within these snapshots indicate that the user should press the <**Enter**> key on the keyboard. Other special keys are represented within **< >** angle brackets. For example, pressing the **PF1** key can be represented as pressing **<PF1>**.
- Author's comments are displayed in italics or as "callout" boxes.

NOTE: Callout boxes refer to labels or descriptions usually enclosed within a box, which point to specific areas of a displayed image.

- This manual refers to the M programming language. Under the 1995 American National Standards Institute (ANSI) standard, M is the primary name of the MUMPS programming language, and MUMPS is considered an alternate name. This manual uses the name M.

- All uppercase is reserved for the representation of M code, variable names, or the formal name of options, field/file names, and security keys (e.g., the XUPROGMODE security key).

  NOTE: Other software code (e.g., Delphi/Pascal and Java) variable names and file/folder names can be written in lower or mixed case.

Documentation Navigation

This document uses Microsoft® Word's built-in navigation for internal hyperlinks. To add **Back** and **Forward** navigation buttons to your toolbar, do the following:

1. Right-click anywhere on the customizable Toolbar in Word 2007 (not the Ribbon section).
2. Select **Customize Quick Access Toolbar** from the secondary menu.
3. Press the drop-down arrow in the "Choose commands from:" box.
4. Select **All Commands** from the displayed list.
5. Scroll through the command list in the left column until you see the **Back** command (green circle with arrow pointing left).
6. Click/Highlight the **Back** command and press **Add** to add it to your customized toolbar.
7. Scroll through the command list in the left column until you see the **Forward** command (green circle with arrow pointing right).
8. Click/Highlight the Forward command and press **Add** to add it to your customized toolbar.
9. Press **OK**.

You can now use these **Back** and **Forward** command buttons in your Toolbar to navigate back and forth in your Word document when clicking on hyperlinks within the document.

NOTE: This is a one-time setup and is automatically available in any other Word document once you install it on the Toolbar.

Commonly Used Terms

The following is a list of terms and their descriptions that you may find helpful while reading the RPC Broker documentation:

Table 3. Commonly used RPC Broker terms

Term	Description
Client	A single term used interchangeably to refer to a user, the workstation (i.e., PC), and the portion of the program that runs on the workstation.
Component	A software object that contains data and code. A component may or may not be visible. **REF:** For a more detailed description, see the *Borland Delphi for Windows User Guide*.
GUI	The Graphical User Interface application that is developed for the client workstation.
Host	The term Host is used interchangeably with the term Server.
Server	The computer where the data and the RPC Broker remote procedure calls (RPCs) reside.

REF: See the "Glossary" for additional terms and definitions.

How to Obtain Technical Information Online

Exported VistA M Server-based software file, routine, and global documentation can be generated using Kernel, MailMan, and VA FileMan utilities.

NOTE: Methods of obtaining specific technical information online are indicated where applicable under the appropriate section.

REF: See the *RPC Broker Technical Manual* for further information.

Help at Prompts

VistA M Server-based software provides online help and commonly used system default prompts. Users are encouraged to enter question marks at any response prompt. At the end of the help display, you are immediately returned to the point from which you started. This is an easy way to learn about any aspect of VistA M Server-based software.

Obtaining Data Dictionary Listings

Technical information about VistA M Server-based files and the fields in files is stored in data dictionaries (DD). You can use the List File Attributes option on the Data Dictionary Utilities submenu in VA FileMan to print formatted data dictionaries.

REF: For details about obtaining data dictionaries and about the formats available, see the "List File Attributes" chapter in the "File Management" section of the *VA FileMan Advanced User Manual*.

Assumptions

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

- VistA computing environment:
  - Kernel—VistA M Server software
  - Remote Procedure Call (RPC) Broker—VistA Client/Server software
  - VA FileMan data structures and terminology—VistA M Server software
- Microsoft Windows environment
- M programming language
- Object Pascal programming language
- Object Pascal programming language/Embarcadero Delphi Integrated Development Environment (IDE)—RPC Broker

References

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

- *RPC Broker Release Notes*
- *RPC Broker Installation Guide*
- *RPC Broker Systems Management Guide*
- *RPC Broker Technical Manual*
- *RPC Broker User Guide* (this manual)

- *RPC Broker Developer's Guide*—Document and BDK Online Help, which provides an overview of development with the RPC Broker. The help is distributed in two zip files:
  - Broker_1_1.zip (i.e., Broker_1_1.chm)—This zip file contains the standalone online HTML help file. Unzip the contents and double-click on the **Broker_1_1.chm** file to open the help.
  - Broker_1_1-HTML_Files.zip—This zip file contains the associated HTML help files. Unzip the contents in the same directory and double-click on the **index.htm** file to open the help.

  You may want to make an entry for **Broker_1_1.chm** in Delphi's Tools Menu, to make it easily accessible from within Delphi. To do this, use Delphi's **Tools | Configure Tools** option and create a new menu entry.
- RPC Broker VA Intranet website.

  This site provides announcements, additional information (e.g., Frequently Asked Questions [FAQs], advisories), documentation links, archives of older documentation and software downloads.

VistA documentation is made available online in Microsoft Word format and in Adobe Acrobat Portable Document Format (PDF). The PDF documents *must* be read using the Adobe Acrobat Reader, which is freely distributed by Adobe Systems Incorporated at the following Website: http://www.adobe.com/

VistA documentation can be downloaded from the VA Software Document Library (VDL) Website: http://www.va.gov/vdl/

VistA documentation and software can also be downloaded from the Health Product Support (HPS) Anonymous Directories.

1 Introduction

The Remote Procedure Call (RPC) Broker (also referred to as "Broker") is a client/server system within Department of Veterans Affairs (VA) 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 VistA M Servers.

This manual provides an overview of software development with the RPC Broker. It introduces developers to the RPC Broker and the Broker Development Kit (BDK) with emphasis on using the RPC Broker in conjunction with Embarcadero's Delphi software. However, the RPC Broker supports other development environments.

REF: For more complete information on development with the RPC Broker components, see the *RPC Broker Developer's Guide* (i.e., BDK Online Help; Broker_1_1.chm).

This document is intended for the VistA development community and Information Resource Management (IRM) staff. A wider audience of technical personnel engaged in operating and maintaining the Department of Veterans Affairs (VA) software may also find it useful as a reference.

1.1 About this Version of the BDK

RPC Broker 1.1 (fully patched) provides developers with the capability to create new VistA client/server software using the following RPC Broker Delphi components in the 32-bit environment:

- TCCOWRPCBroker
- TContextorControl
- TRPCBroker (original component)
- TSharedBroker
- TSharedRPCBroker
- TXWBRichEdit

NOTE: These RPC Broker components wrap the functionality of the Broker resulting in a more modularized and orderly interface. Those components derived from the original TRPCBroker component, inherit the TRPCBroker properties and methods.

1.1.1 Features

This enhanced Broker software has the following functionality/features:

- Supports Secure Shell (SSH)—As of RPC Broker Patch XWB*1.1*50, the TRPCBroker component enabled Secure Shell (SSH) Tunnels to be used for secure connections. This functionality is controlled by setting an internal property value (mandatory SSH) or command line option at run time.
- Supports Broker Security Enhancement (BSE)—As of RPC Broker Patch XWB*1.1*50, the TRPCBroker component enabled visitor access to remote sites using authentication established at a home site.
- Supports Single Sign-On/User context (SSO/UC)—TCCOWRPCBroker component enables Single Sign-On/User Context (SSO/UC) in CCOW-enabled applications.
- Supports Non-Callback Connections—RPC Broker components are built with a UCX or non-callback Broker connection, so that it can be used from behind firewalls, routers, etc. This functionality is controlled via the new TRPCBroker component IsBackwardCompatibleConnection property.
- Supports Silent Logon capabilities—RPC Broker provides "Silent Login" capability. It provides functionality associated with the ability to make logins to a VistA M Server without the RPC Broker asking for Access and Verify code information.
- Documented Deferred RPCs and Capability to Run RPCs on a Remote Server.
- Multi-instances of the RPC Broker—RPC Broker code permits an application to open two separate Broker instances with the same Server/ListenerPort combination, resulting in two separate partitions on the server. Previously, an attempt to open a second Broker instance ended up using the same partition. For this capability to be useful for concurrent processing, an application would have to use threads to handle the separate Broker sessions.

CAUTION: Although we believe there should be no problems, the RPC Broker is not yet guaranteed to be thread safe.

- Updated components, properties, methods, and types.
- Separate Design-time and Run-time Packages—BDK contains separate run-time and design-time packages.
- Supports Delphi XE5, XE4, XE3, and XE2.

To develop VistA applications in a 32-bit environment you must have Delphi XE2 or greater. RPC Broker 1.1 does *not* allow you to develop applications in Delphi 1.0. However, the Broker routines on the VistA M Server continue to support VistA applications previously developed in the 16-bit environment.

The default installation of the Broker creates a separate BDK directory (i.e., BDK32) that contains the required Broker files for development.

1.1.2 Backward Compatibility Issues

Client applications compiled with RPC Broker 1.1 do *not* work at a site that has not upgraded its server software to RPC Broker 1.1.

On the other hand, client applications compiled with RPC Broker 1.0 do work with the RPC Broker 1.1 server software.

2 RPC Broker Components for Delphi

REF: For more detailed information on the RPC Broker components for Delphi, see the *RPC Broker Developer's Guide* (i.e., BDK Online Help; Broker_1_1.chm).

2.1 TRPCBroker Component

The main tool to develop client applications for the RPC Broker environment is the TRPCBroker component for Delphi. The TRPCBroker component adds the following abilities to your Delphi application:

- Connecting to an VistA M Server:
    - Authenticate the user.
    - Set up the environment on the VistA M Server.
    - Bring back the introductory text.
- Invoking Remote Procedure Calls (RPCs) on the VistA M Server:
    - Send data to the VistA M Server.
    - Perform actions on the VistA M Server.
    - Return data from the VistA M Server to the client.

To add the TRPCBroker component to your Delphi application, simply drop it from the **Kernel** tab of Delphi's component palette to a form in your application.

2.1.1 TRPCBroker Properties and Methods

As a Delphi component, the TRPCBroker component is controlled and accessed through its properties and methods. By setting its properties and executing its methods, you can connect to a VistA M Server from your application and execute RPCs on the VistA M Server to exchange data and perform actions on the VistA M Server.

For most applications, you only need to use a single TRPCBroker component to manage communications with the VistA M Server.

2.1.2 TRPCBroker Key Properties

The following table lists the most important properties of the TRPCBroker component.

REF: For a complete list of all of Broker properties, see the *RPC Broker Developer's Guide* (i.e., BDK Online Help; Broker_1_1.chm).

Table 4: TRPCBroker component key properties

Property	Description
ClearParameters	If **True**, the Param property is cleared *after* every invocation of the Call, strCall, or the lstCall methods.
ClearResults	If **True,** the Results property is cleared *before* every invocation of the Call method, thus assuring that only the results of the last call are returned.
Connected	Setting this property to **True** connects your application to the server.
ListenerPort	Sets server port to connect to a Broker Listener process (mainly for development purposes; for end-users, determine on the fly with GetServerInfo method.)
Param	Run-time array in which you set any parameters to pass as input parameters when calling an RPC on the server.
RemoteProcedure	Name of a RemoteProcedure entry that the Call, lstCall, or strCall method should invoke.
Results	This is where any results are stored after a Call, lstCall, or strCall method completes.
Server	Name of the server to connect to (mainly for development purposes; for end-users, determine on the fly with GetServerInfo method.)

2.1.3 TRPCBroker Key Methods

This section lists the most important methods of the TRPCBroker component.

REF: For a complete list of all of Broker methods, see the *RPC Broker Developer's Guide* (i.e., BDK Online Help; Broker_1_1.chm).

Table 5. TRPCBroker component methods

Method	Description
procedure Call;	This method executes an RPC on the server and returns the results in the TRPCBroker component's Results property. Call expects the name of the remote procedure and its parameters to be set up in the RemoteProcedure and Param properties respectively. If ClearResults is True, then the Results property is cleared before the call. If ClearParameters is True, then the Param property is cleared after the call finishes.
function strCall: string;	This method is a variation of the Call method. Only use it when the return type is a single string. Instead of returning results in the TRPCBroker component's Results[0] property node, results are returned as the value of the function call. Unlike the Call method, the Results property is not affected; no matter the setting of ClearResults, the value is left unchanged.
procedure lstCall(OutputBuffer: TStrings);	This method is a variation of the Call method. Instead of returning results in the TRPCBroker component's Results property, it instead returns results in the TStrings object you specify. Unlike the Call method, the Results property is not affected; no matter the setting of ClearResults, the value is left unchanged.
function CreateContext(strContext: string): boolean;	This method creates a context for your application. Pass an option name in the strContext parameter. If the function returns True, a context was created, and your application can use all RPCs entered in the option's RPC multiple.

Examples

For examples of how to use these methods to invoke RPCs, see the "How to Execute an RPC from a Client Application" section in the "Remote Procedure Calls (RPCs)" chapter.

2.1.4 How to Connect to a VistA M Server

To establish a connection from your application to a Broker on the VistA M Server, perform the following procedure:

1. From the **Kernel** component palette tab, add a TRPCBroker component to your form.
2. Add code to your application to connect to the server; one likely location is your form's OnCreate event handler. The code should:
   a. Use the GetServerInfo function to retrieve the run-time server and port to connect to. This function is not a method of the TRPCBroker component; it is described in the Other RPC Broker APIs chapter.
   b. Inside of an exception handler **try...except** block, set RPCBroker1's Connected property to True. This causes an attempt to connect to the Broker server.
   c. Check if an EBrokerError exception is raised. If this happens, connection failed. You should inform the user of this and then terminate the application.

   The code, placed in an OnCreate event handler, should look like:

   Figure 1: OnCreate event handler—Sample code

```
procedure TForm1.FormCreate(Sender: TObject);
var   ServerStr: String;
        PortStr: String;
begin
  // get the correct port and server from registry
  if GetServerInfo(ServerStr,PortStr)<>mrCancel then
  begin
    RPCBroker1.Server:=ServerStr;
    RPCBroker1.ListenerPort:=StrToInt(PortStr);
  end
  else Application.Terminate;

  // establish a connection to the Broker
  try
    RPCBroker1.Connected:=True;
  except
    On EBrokerError do
    begin
      ShowMessage('Connection to server could not be established!');
      Application.Terminate;
    end;
  end;
end;
```

3. A connection with the Broker on the VistA M Server is now established. You can use the CreateContext method of the TRPCBroker component to authorize use of RPCs for your user, and then use the Call, lstCall, and strCall methods of the TRPCBroker component to execute RPCs on the VistA M Server.

REF: For information on creating and executing RPCs, see the "Remote Procedure Calls (RPCs)" chapter.

2.2 TCCOWRPCBroker Component

As of Patch XWB*1.1*40, the TCCOWRPCBroker component was added to RPC Broker 1.1. The TCCOWRPCBroker Delphi component allows VistA application developers to make their applications CCOW-enabled and Single Sign-On/User Context (SSO/UC)-aware with all of the client/server-related functionality in one integrated component. Using the TCCOWRPCBroker component, an application can share User Context stored in the CCOW Context Vault.

Thus, when a VistA CCOW-enabled application is recompiled with the TCCOWRPCBroker component and other required code modifications are made, that application would then become SSO/UC-aware and capable of single sign-on (SSO).

NOTE: This RPC Broker component is derived from the original TRPCBroker Component; it inherits the TRPCBroker properties and methods.

2.2.1 Single Signon/User Context (SSO/UC)

The Veterans Health Administration (VHA) information systems user community expressed a need for a single sign-on (SSO) service with interfaces to VistA, Health*e*Vet VistA, and non-VistA systems. This architecture allows users to authenticate and sign on to multiple applications that are CCOW-enabled and SSO/UC-aware using a single set of credentials, which reduces the need for multiple ID's and passwords in the Health*e*Vet clinician desktop environment. The RPC Broker software addressed this architectural need by providing a new TCCOWRPCBroker component in RPC Broker Patch XWB*1.1*40.

The TCCOWRPCBroker component allows VistA application developers to make their applications CCOW-enabled and Single Sign-On/User Context (SSO/UC)-aware with all of the client/server-related functionality in one integrated component. Using the TCCOWRPCBroker component, an application can share User Context stored in the CCOW Context Vault.

Thus, when a VistA CCOW-enabled application is recompiled with the TCCOWRPCBroker component and other required code modifications are made, that application would then become SSO/UC-aware and capable of single sign-on (SSO).

REF: For more information on SSO/UC and making your Broker-based applications CCOW-enabled and SSO/UC-aware, please consult the *Single Sign-On/User Context (SSO/UC) Installation Guide* and *Single Sign-On/User Context (SSO/UC) Deployment Guide* on the VHA Software Documentation Library (VDL).

2.3 TContextorControl Component

As of RPC Broker Patch XWB*1.1*40, the TContextorControl component was added to RPC Broker 1.1. The TContextorControl Delphi component communicates with the Vergence Locator service.

2.4 TSharedBroker Component

As of Patch XWB*1.1*26, the TSharedBroker component was added to RPC Broker 1.1. The TSharedBroker Delphi component provides applications or plugins to applications easy access to an RPC Broker component without the need for a separate M partition. Each component has its own security (i.e., option) as well. The default value of the AllowShared property is **True**. If an application has RPCs that require extensive time, it would be best to *not* share a Broker instance and the AllowShared property should then be set to **False**.

NOTE: This RPC Broker component is derived from the original TRPCBroker Component; it inherits the TRPCBroker properties and methods.

2.5 TSharedRPCBroker Component

As of Patch XWB*1.1*26, the TSharedBroker component was added to RPC Broker 1.1. The TSharedRPCBroker Delphi component provides applications or plugins to applications easy access to RPC Broker components without the need for a separate M partition. Each component has its own security (i.e., option) as well. The default value of the AllowShared is **True**. If an application has RPCs that require extensive time, it would be best to *not* share a Broker instance and the AllowShared property should then be set to **False**.

NOTE: This RPC Broker component is derived from the original TRPCBroker Component; it inherits the TRPCBroker properties and methods.

2.6 TXWBRichEdit Component

As of Patch XWB*1.1*13, the TXWBRichEdit component was added to RPC Broker 1.1. The TXWBRichEdit Delphi component replaces the Introductory Text Memo component on the Login Form. TXWBRichEdit is a version of the TRichEdit component that uses Version 2 of Microsoft's® RichEdit Control and adds the ability to detect and respond to a Uniform Resource Locator (URL) in the text. This component permits us to provide some requested functionality on the login form. As an XWB namespaced component we are required to put it on the **Kernel** tab of the component palette, however, it rightly belongs on the **Win32** tab.

3 Remote Procedure Calls (RPCs)

3.1 What is a Remote Procedure Call?

A remote procedure call (RPC) is a defined call to M code that runs on a VistA M Server. A client application, through the RPC Broker, can make a call to the VistA M Server and execute an RPC on the VistA M Server. This is the mechanism through which a client application can:

- Send data to a VistA M Server.
- Execute code on a VistA M Server.
- Retrieve data from a VistA M Server.

An RPC can take optional parameters to do some task and then return either a single value or an array to the client application. RPCs are stored in the REMOTE PROCEDURE file (#8994).

3.1.1 Relationship between an M Entry Point and an RPC

An RPC can be thought of as a wrapper placed around an M entry point for use with client applications. Each RPC invokes a single M entry point. The RPC passes data in specific ways to its corresponding M entry point and expects any return values from the M entry point to be returned in a pre-determined format. This allows client applications to connect to the RPC Broker, invoke an RPC, and through the RPC, invoke an M entry point on a server.

3.2 Create Your Own RPCs

You can create your own custom RPCs to perform actions on the VistA M Server and to retrieve data from the VistA M Server. Then you can call these RPCs from your client application. Creating an RPC requires you to perform the following two steps:

1. Write and test the M entry point that is called by the RPC.
2. Add the RPC entry that invokes your M entry point, in the REMOTE PROCEDURE file (#8994).

3.3 Writing M Entry Points for RPCs

3.3.1 First Input Parameter for RPCs (Required)

The RPC Broker always passes a variable by reference in the first input parameter to your M routine. It expects results (one of five types described in Table 6) to be returned in this parameter. You must always set some return value into that first parameter before your routine returns.

3.3.2 Return Value Types for RPCs

There are five RETURN VALUE TYPES for RPCs as shown in Table 6. Choose a return value type that is appropriate to the type of data your RPC needs to return to your client. Your M entry point should set the return value (in the routine's first input parameter) accordingly.

Table 6: RPC Broker return value types

RPC Return Value Type	How M Entry Point Should Set the Return Parameter	RPC WORD WRAP ON Setting	Value(s) returned in Client Results

Single Value	Set the return parameter to a single value.
	For example:
	`TAG(RESULT) ;`
	` S RESULT="DOE, JOHN"`
	` Q`
		No effect	Value of parameter, in Results[0].

Array	Set an array of strings into the return parameter, each subscripted one level descendant.
	For example:
	`TAG(RESULT) ;`
	` S RESULT(1)="ONE"`
	` S RESULT(2)="TWO"`
	` Q`
	For large arrays consider using the GLOBAL ARRAY return value type to avoid memory allocation errors.
		No effect	Array values, each in a Results item.

Word-processing	Set the return parameter the same as you set it for the ARRAY type. The only difference is that the WORD WRAP ON field (#.08) setting affects the Word-processing return value type.	True	Array values, each in a Results item.
		False	Array values, concatenated into Results[0].
Global Array	Set the return parameter to a closed global reference in ^TMP. The global's data nodes are traversed using $QUERY, and all data values on global nodes descendant from	True	Array values, each in a Results item.
		False	Array values,

RPC Return Value Type	How M Entry Point Should Set the Return Parameter	RPC WORD WRAP ON Setting	Value(s) returned in Client Results

	the global reference are returned.
	This type is especially useful for returning data from VA FileMan word processing fields, where each line is on a 0-subscripted node.
	CAUTION: The global reference you pass is killed by the Broker at the end of RPC Execution as part of RPC cleanup. Do not pass a global reference that is not in ^TMP or that should not be killed.
	This type is useful for returning large amounts of data to the client, where using the ARRAY type can exceed the symbol table limit and crash your RPC.
	For example, to return signon introductory text you could do:
	`TAG(RESULT);`
	` M ^TMP("A6A",$J)=`
	`^XTV(8989.3,1,"INTRO")`
	` ;this node not needed`
	` K ^TMP("A6A",$J,0)`
	` S RESULT=$NA(^TMP("A6A",$J))`
	` Q`
			concatenated into Results[0].

Global Instance	Set the return parameter to a closed global reference.
	For example, to return the 0th node from the NEW PERSON file (#200) for the current user:
	`TAG(RESULT) ;`
	` S RESULT=$NA(^VA(200,DUZ,0))`
	` Q`
		No effect	Value of global node, in Results[0].

3.3.3 Input Parameter Types for RPCs (Optional)

The M entry point for an RPC can optionally have input parameters (i.e., beyond the first parameter, which is always used to return an output value). The client passes data to your M entry point through these parameters.

The client can send data to an RPC (and therefore your entry point) in one of the following type format types:

Table 7: Input parameter types—PType Property Values

Value	Definition
literal	Delphi string value, passed as a string literal to the VistA M Server. The VistA M Server receives the contents of the corresponding Value property as one string or one number.
reference	Delphi string value, treated on the VistA M Server as an M variable name and resolved from the symbol table at the time the RPC executes. The VistA M Server receives the contents of the corresponding Value property as a name of a variable defined on the server. Using indirection, the Broker on the server resolves this parameter before handing it off to the application.
list	A single-dimensional array of strings in the Mult subproperty of the Param property, passed to the VistA M Server where it is placed in an array. String subscripting can be used. This value is used whenever an application wants to send a list of values to the VistA M Server. Data is placed in a local array. In this case, the contents of the corresponding Mult property is sent, while the Value property is ignored.
global	This value is similar to list, but instead of data being placed in a local array, it is placed in a global array. Use of this value removes the potential problem of allocation errors when large quantities of data are transmitted. This value was made available as of RPC Broker Patch XWB*1.1*40.
empty	This value indicates that no parameter value is to be passed; it simply passes an empty argument. This value was made available as of RPC Broker Patch XWB*1.1*40.
stream	This value indicates that the data should be passed as a single stream of data. This value was made available as of RPC Broker Patch XWB*1.1*40.
undefined	The Broker uses this value internally. It should *not* be used by an application.

The type of the input parameters passed in the Param property of the TRPCBroker component determines the format of the data you must be prepared to receive in your M entry point.

3.3.4 RPC M Entry Point Examples

The following two examples illustrate sample M code that could be used in simple RPCs.

3.3.4.1 Sum of Two Numbers

The following example takes two numbers and returns their sum:

Figure 2: RPC M entry point example—Sum of two numbers

```
SUM(RESULT,A,B)    ;add two numbers
 S RESULT=A+B
 Q
```

3.3.4.2 Sorted Array

The following example receives an array of numbers and returns them as a sorted array to the client:

Figure 3: RPC M entry point example—Sorted array

```
SORT(RESULT,UNSORTED)    ;sort numbers
 N I
 S I=""
 F  S I=$O(UNSORTED(I)) Q:I=""  S RESULT(UNSORTED(I))=UNSORTED(I)
 Q
```

3.4 RPC Entry in the REMOTE PROCEDURE File

After the M code is complete, you need to create the RPC itself in the REMOTE PROCEDURE file (#8994). The following fields in the REMOTE PROCEDURE file (#8994) are key to the correct operation of an RPC:

Table 8: REMOTE PROCEDURE file key field entries

Field Name	Required?	Description
NAME (#.01)	Yes	The name that identifies the RPC (this entry should be namespaced in the package namespace).
TAG (#.02)	Yes	The tag at which the remote procedure call begins.
ROUTINE (#.03))	Yes	The name of the routine that should be invoked to start the RPC.
WORD WRAP ON (#.08)	No	Affects Global Array and Word-processing return value types only. If set to **False**, data is returned in a single concatenated string in Results[0]. If set to **True**, each array node on the M side is returned as a

Field Name	Required?	Description
		distinct array item in Results.
RETURN VALUE TYPE (#.04)	Yes	This indicates to the Broker how to format the return values. For example, if the RETURN VALUE TYPE is set as Word-processing, then each entry in the returning list has a <CR><LF> (<carriage return><line feed>) appended.

3.5 What Makes a Good Remote Procedure Call?

- Silent calls (no I/O to terminal or screen, no user intervention required).
- Minimal resources required (passes data in brief, controlled increments).
- Discrete calls (requiring as little information as possible from the process environment).
- Generic as possible (different parts of the same package as well as other packages could use the same RPC).

3.6 How to Execute an RPC from a Client Application

To execute an RPC from a client application, perform the following procedure:

1. If your RPC has any input parameters beyond the mandatory first parameter, set a Param node in the TRPCBroker's Param property for each. For each input parameter, set the following sub properties:

    - Value
    - PType (Literal, List, or Reference).

    If the parameter's PType is List, however, set a list of values in the Mult subproperty rather than setting the Value subproperty.

    Figure 4 is an example of some settings of the Param property:

    Figure 4: Param property—Sample settings

    ```
    RPCBroker1.Param[0].Value := '10/31/97';
    RPCBroker1.Param[0].PType := literal;
    RPCBroker1.Param[1].Mult['"NAME"'] := 'SMITH, JOHN';
    RPCBroker1.Param[1].Mult['"SSN"'] := '123-45-6789';
    RPCBroker1.Param[1].PType := list;
    ```

2. Set the TRPCBroker's RemoteProcedure property to the name of the RPC to execute.

    ```
    RPCBroker1.RemoteProcedure:='A6A LIST';
    ```

3. Invoke the Call method of the TRPCBroker component to execute the RPC. All calls to the Call method should be done within an exception handler try...except statement, so that all communication errors (which trigger the EBrokerError exception) can be trapped and handled. For example:

Figure 5: Exception handler—try...except code—Sample usage

```
try
    RPCBroker1.Call;
except
  On EBrokerError do
    ShowMessage('A problem was encountered communicating with the server.');
end;
```

4. Any results returned by your RPC are returned in the TRPCBroker component's Results property. Depending on how you set up your RPC, results are returned either in a single node of the Results property (Result[0]) or in multiple nodes of the Results property.

NOTE: You can also use the lstCall and strCall methods to execute an RPC. The main difference between these methods and the Call method is that lstCall and strCall do not use the Results property, instead returning results into a location you specify.

3.7 RPC Security: How to Register an RPC

Security for RPCs is handled through the RPC registration process. Each client application must create a context for itself, which checks if the application user has access to a "B"-type option in the Kernel menu system. Only RPCs assigned to that option can be run by the client application.

To enable your application to create a context for itself, perform the following procedure:

1. Create a "B"-type option in the OPTION file (#19) for your application.

NOTE: The OPTION TYPE "**B**" represents a **B**roker client/server type option.

2. In the RPC multiple for this option type, add an entry for each RPC that your application calls. You can also specify a security key that can lock each RPC (this is a pointer to the SECURITY KEY file [#19.1]) and M code in the RULES subfield that can also determine whether to enable access to each RPC.
3. When you export your software using KIDS, export both your RPCs and your software option.
4. Your application must create a context for itself on the server, which checks access to RPCs. In the initial code of your client application, make a call to the CreateContext method of your TRPCBroker component. Pass your application's "B"-type option's name as a parameter. For example:

```
RPCBroker1.CreateContext(option_name)
```

If the CreateContext method returns **True**, only those RPCs designated in the RPC multiple of your application option is permitted to run.

If the CreateContext method returns **False**, you should terminate your application (if you do not your application runs, but you get errors every time you try to access an RPC).

5. End-users of your application must have the "B"-type option assigned to them on one of their menus, in order for the CreateContext method to return True.

3.7.1 Bypassing RPC Security for Development

Having the XUPROGMODE security key allows you to bypass the Broker security checks. You can run any RPC without regard to application context (without having to use the CreateContext method). This is a convenience for application development. When you complete development, make sure you test your application from an account *without* the XUPROGMODE key, to ensure that all RPCs needed are properly registered.

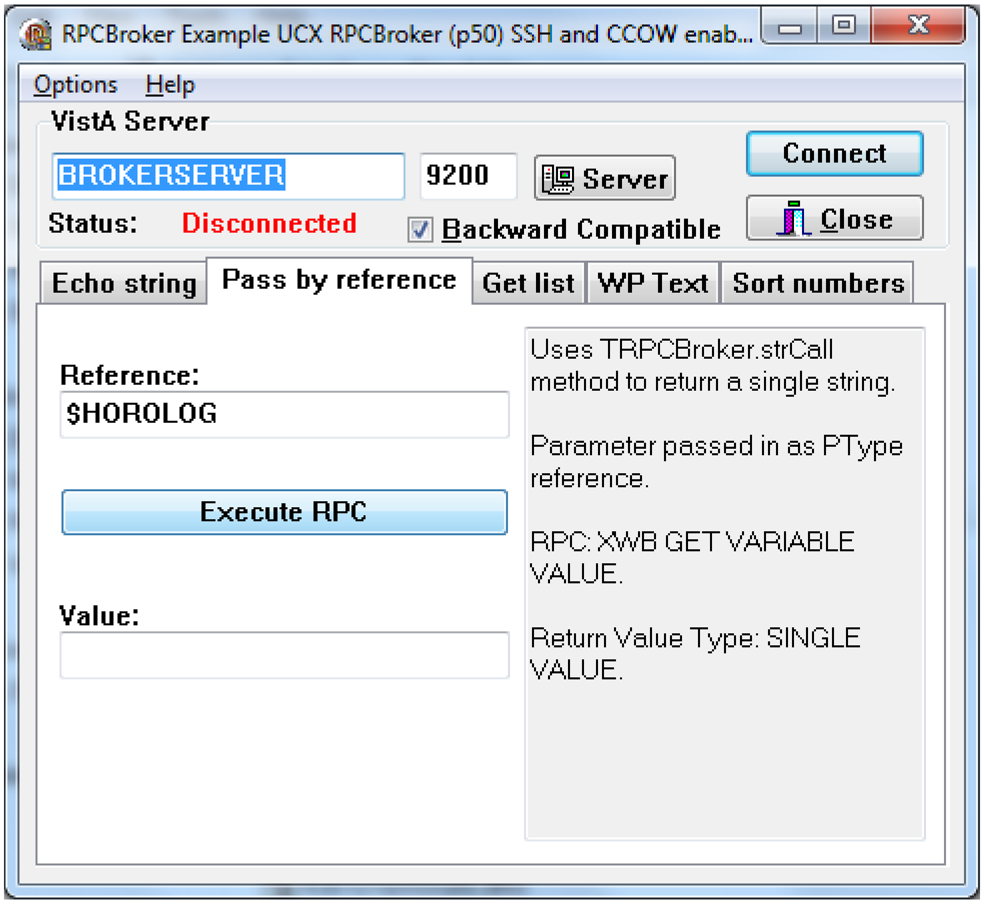
3.7.2 BrokerExample Online Code Example

The BrokerExample sample application (i.e., BROKEREXAMPLE.EXE) provided with the BDK demonstrates the basic features of developing RPC Broker-based applications, including:

- Connecting to a VistA M Server.
- Creating an application context.
- Using the GetServerInfo function.
- Displaying the VistA splash screen.
- Setting the TRPCBroker Param property for each Param PType (literal, reference, and list).
- Calling RPCs with the Call method.
- Calling RPCs with the lstCall and strCall methods.

The client source code files for the BrokerExample application are located in the SAMPLES\RPCBROKER\BROKEREX subdirectory of the main BDK32 directory.

Figure 6: RPCBroker Example application



Uses TRPCBroker.strCall method to return a single string.

Parameter passed in as PType reference.

RPC: XWB GET VARIABLE VALUE.

Return Value Type: SINGLE VALUE.

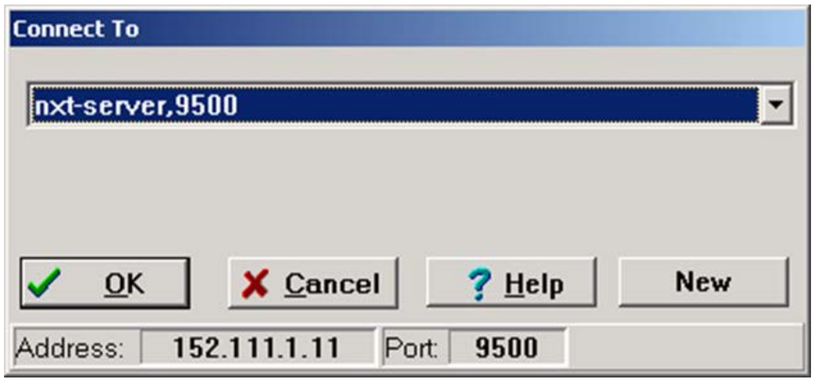
4 Other RPC Broker APIs

4.1 GetServerInfo Function

The GetServerInfo function retrieves the end-user workstation's server and port. Use this function to set the TRPCBroker component's Server and ListenerPort properties to reflect the end-user workstation's settings before connecting to the server.

If there is more than one server/port to choose from, GetServerInfo displays dialogue that allows users to select a service to connect to, as shown in Figure 7:

Figure 7: Server and port configuration selection dialogue


Connect To

nxt-server,9500

Address: 152.111.1.11 Port: 9500

If exactly one server and port entry is defined in the Microsoft Windows Registry, GetServerInfo does *not* display this dialogue. The values in the single Microsoft Windows Registry entry are returned, with no user interaction required.

If more than one server and port entry exists in the Microsoft Windows Registry, the dialogue is displayed, and the user chooses to which server they want to connect.

If no values for server and port are defined in the Microsoft Windows Registry, GetServerInfo does not display this dialogue, and automatic default values are returned (i.e., BROKERSERVER and 9200).

Syntax of GetServerInfo Function:

```
function GetServerInfo(var Server, Port: string): integer;
```

NOTE: The unit is RpcConf1.

4.2 VistA Splash Screen Procedures

Two procedures in SplVista.PAS unit are provided to display a VistA splash screen when an application loads:

- **`procedure`** `SplashOpen;`
- **`procedure`** `SplashClose(TimeOut:` **`longint`**`);`

It is recommended that the splash screen be opened and closed in the section of Pascal code in an application's project file (i.e., .DPR).

To use the splash screen in an application, perform the following procedure:

1. Open your application's project (**.DPR**) file (in Delphi, choose View | Project Source).
2. Include the SplVista in the uses clause of the project source.
3. Call SplashOpen immediately after the first form of your application is created and call SplashClose just prior to invoking the Application.Run method.
4. Use the TimeOut parameter to ensure a minimum display time.

Figure 8: VistA Splash screen

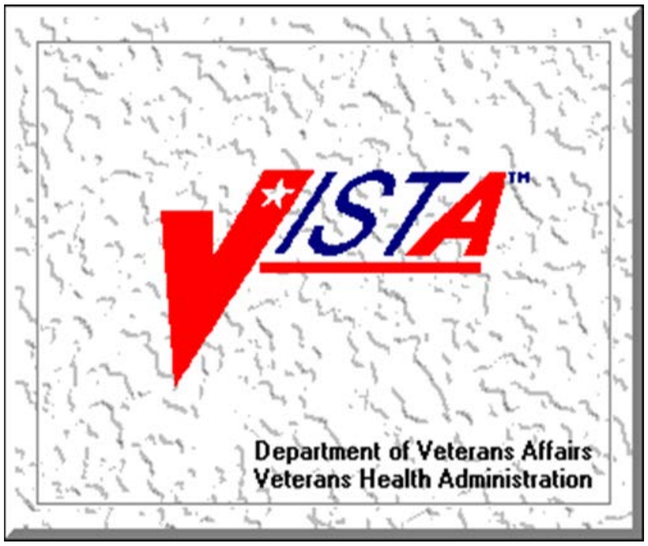

VISTA™
Department of Veterans Affairs
Veterans Health Administration

Figure 9: Displaying a VistA splash screen: Sample code

```
uses
  Forms, Unit1 in 'Unit1.pas', SplVista;

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  SplashOpen;
  SplashClose(2000);
  Application.Run;
end.
```

4.3 XWB GET VARIABLE VALUE RPC

You can call the XWB GET VARIABLE VALUE RPC (distributed with the RPC Broker) to retrieve the value of any M variable in the server environment. Pass the variable name in Param[0].Value and the type (reference) in Param[0].PType. Also, the current context of your user must give them permission to execute the XWB GET VARIABLE VALUE RPC (it must be included in the RPC multiple of the "B"-type option registered with the CreateContext function). For example:

Figure 10: XWB GET VARIABLE VALUE RPC usage—Sample code

```
RPCBroker1.RemoteProcedure := 'XWB GET VARIABLE VALUE';
RPCBroker1.Param[0].Value :='DUZ';
RPCBroker1.Param[0].PType := reference;
try
    RPCBroker1.Call;
except
  On EBrokerError do
    ShowMessage('Connection to server could not be established!');
end;
ShowMessage('DUZ is '+RPCBroker1.Results[0]);
```

4.4 M Emulation Functions

Piece Function

The Piece function is a scaled down Pascal version of M's $PIECE function. It is declared in MFUNSTR.PAS.

```
function Piece(x: string; del: string; piece: integer) : string;
```

4.4.1 Translate Function

The Translate function is a scaled down Pascal version of M's $TRANSLATE function. It is declared in MFUNSTR.PAS.

```
function Translate(passedString, identifier, associator: string): string;
```

4.5 Encryption Functions

Kernel and the RPC Broker provide some rudimentary encryption and decryption functions. Data can be encrypted on the client end and decrypted on the server, and vice-versa.

4.5.1 In Delphi

Include HASH in the "uses" clause of the unit in which you'll be encrypting or decrypting.

Function prototypes are as follows:

- `function Decrypt(EncryptedText: string): string;`
- `function Encrypt(NormalText: string): string;`

4.5.2 On the VistA M Server

4.5.2.1 Encryption

To encrypt:

Figure 11: Encryption in VistA M Server—Sample code

```
>S CIPHER=$$ENCRYP^XUSRB1("Hello world!") W CIPHER

/U'llTG~TVl&f-
```

4.5.2.2 Decryption

To decrypt:

Figure 12: Decryption in VistA M Server—Sample code

```
>S PLAIN=$$DECRYP^XUSRB1(CIPHER) W PLAIN

Hello world!
```

4.6 $$BROKER^XWBLIB

Use this function in the M code called by an RPC to determine if the Broker is executing the current process. It returns 1 if this is true, 0 if false.

4.7 $$RTRNFMT^XWBLIB

Use this function in the M code called by an RPC to change the return value type that the RPC returns on-the-fly. This allows you to change the return value type to any valid return value type (Single Value, Array, Word-processing, Global Array, or Global Instance). It also lets you set WORD WRAP ON to True or False, on-the-fly, for the RPC.

REF: For more information about $$RTRNFMT^XWBLIB, see the *RPC Broker Developer's Guide* (i.e., BDK Online Help; Broker_1_1.chm).

5 Debugging and Troubleshooting

5.1 How to Debug Your Client Application

Beside the normal debugging facilities provided by Delphi, you can also invoke a debug mode, so that you can step through your code on the client side and your RPC code on the VistA M Server side simultaneously.

To invoke the debug mode, perform the following procedure:

1. On the client side, set the DebugMode property on the TRPCBroker component to True. When the TRPCBroker component connects with this property set to **True**, you get a dialogue indicating your workstation IP address and the port number.
2. At this point, switch over to the VistA M Server and set any break points in the routines being called in order to help isolate the problem. Then issue the M debug command (e.g., ZDEBUG in DSM).
3. Start the following VistA M Server process:

   ```
   >D EN^XWBTCP
   ```

   You are prompted for the workstation IP address and the port number. After entering the information, switch over to the client application and click **OK**.
4. You can now step through the code on your client and simultaneously step through the code on the server side for any RPCs that your client calls.

5.1.1 RPC Error Trapping

M errors on the VistA M Server that occur during RPC execution are trapped by the use of M and Kernel error handling. In addition, the M error message is sent back to the Delphi client. Delphi raises an exception EBrokerError and displays a popup box with the error. At this point, RPC execution terminates and the channel is closed.

5.2 Troubleshooting Connections

5.2.1 Identifying the Listener Process on the Server

On DSM systems, where the Broker Listener is running, the Listener process name is RPCB_Port:*####*, where *####* is the port number being listened to. This should help quickly locate Listener processes when troubleshooting any connection problems.

5.2.2 Identifying the Handler Process on the Server

On DSM systems the name of a Handler process is ip*XXX.XXX*: *####*, where *XXX.XXX* is the last two octets of the client IP address and *####* is the port number.

5.2.3 Testing Your RPC Broker Connection

To test the RPC Broker connection from your workstation to the VistA M Server, use the RPC Broker Diagnostic Program (RPCTEST.EXE).

REF: For a complete description of the RPC Broker Diagnostic program, see the "Troubleshooting" chapter in the *RPC Broker Systems Management Guide*.

6 RPC Broker and Delphi

The following sections highlight changes made to or comments about the RPC Broker to accommodate a particular version of Delphi. They are listed in reverse Delphi version order.

RECOMMENDATION: To avoid problems with the BDK, it is *recommended* for all Delphi packages that you accept the default directory after compiling the Broker Development Kit (BDK) on a workstation.

6.1 Delphi XE5 Packages

6.1.1 Delphi XE5 *Starter* Edition—*Not* Recommended for BDK Development

Delphi XE5 comes in three flavors:

- Starter
- Professional
- Enterprise

RECOMMENDATION: It is *recommended* that you use either the Professional or Enterprise version of Delphi XE5 to develop applications using the RPC Broker.

REF: For more information on the different editions of Delphi, see the "Delphi XE2 Starter Edition—Not Recommended for BDK Development" section.

6.1.2 XWB_RXE5.bpl File

This run-time package contains the source code for the standard RPCBroker components and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation:

C:\Users\Public\Documents\RAD Studio\12.0

6.1.3 XWB_DXE5.bpl File

This design-time package contains the installed components for the standard RPCBroker and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation:

C:\Users\Public\Documents\RAD Studio\12.0

6.1.4 SharedRPCBroker_RXE5.bpl File

This run-time package contains the source code for the SharedRPCBroker components and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation:

C:\Users\Public\Documents\RAD Studio\12.0

6.1.5 SharedRPCBroker_DXE5.bpl File

This design-time package contains the installed components for the SharedRPCBroker and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation:

C:\Users\Public\Documents\RAD Studio\12.0

6.2 Delphi XE4 Packages

6.2.1 Delphi XE4 *Starter* Edition—*Not* Recommended for BDK Development

Delphi XE4 comes in three flavors:

- Starter
- Professional
- Enterprise

RECOMMENDATION: It is *recommended* that you use either the Professional or Enterprise version of Delphi XE4 to develop applications using the RPC Broker.

REF: For more information on the different editions of Delphi, see the "Delphi XE2 Starter Edition—Not Recommended for BDK Development" section.

6.2.2 XWB_RXE4.bpl File

This run-time package contains the source code for the standard RPCBroker components and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation:

C:\Users\Public\Documents\RAD Studio\11.0

6.2.3 XWB_DXE4.bpl File

This design-time package contains the installed components for the standard RPCBroker and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation:

C:\Users\Public\Documents\RAD Studio\11.0

6.2.4 SharedRPCBroker_RXE4.bpl File

This run-time package contains the source code for the shared RPCBroker components and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation:

C:\Users\Public\Documents\RAD Studio\11.0

6.2.5 SharedRPCBroker_DXE4.bpl File

This design-time package contains the installed components for the SharedRPCBroker and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation:

C:\Users\Public\Documents\RAD Studio\11.0

6.3 Delphi XE3 Packages

6.3.1 Delphi XE3 *Starter* Edition—*Not* Recommended for BDK Development

Delphi XE3 comes in three flavors:

- Starter
- Professional
- Enterprise

RECOMMENDATION: It is *recommended* that you use either the Professional or Enterprise version of Delphi XE3 to develop applications using the RPC Broker.

REF: For more information on the different editions of Delphi, see the "Delphi XE2 Starter Edition—Not Recommended for BDK Development" section.

6.3.2 XWB_RXE3.bpl File

This run-time package contains the source code for the standard RPCBroker components and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation:

C:\Users\Public\Documents\RAD Studio\10.0

6.3.3 XWB_DXE3.bpl File

This design-time package contains the installed components for the standard RPCBroker and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation:

C:\Users\Public\Documents\RAD Studio\10.0

6.3.4 SharedRPCBroker_RXE3.bpl File

This run-time package contains the source code for the SharedRPCBroker components and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation:

C:\Users\Public\Documents\RAD Studio\10.0

6.3.5 SharedRPCBroker_DXE3.bpl File

This design-time package contains the installed components for the SharedRPCBroker and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation:

C:\Users\Public\Documents\RAD Studio\10.0

6.4 Delphi XE2 Packages

6.4.1 Delphi XE2 *Starter* Edition—*Not* Recommended for BDK Development

Delphi XE2 comes in three flavors:

- Starter
- Professional
- Enterprise

This version of the BDK requires the Professional or Enterprise Edition. The Starter edition is targeted mainly at students, and as such, leaves out many features. We do *not* recommend using the Starter edition of Delphi XE2 for RPC Broker development at this time. Delphi XE2 Starter Edition does *not* ship the following:

- OpenHelp help system—Allow easy integration of 3rd party component help with Delphi's own internal component help.
- VCL source code unit (i.e., "dsgnintf.pas" file)—RPCBroker component has a dependency on a VCL source code unit. Delphi XE2 Starter Edition does *not* ship VCL source code unit in either .PAS or .DCU form; however, VCL Source code units are available in Delphi XE2 Professional and Enterprise editions.

NOTE: When installing Delphi XE2 Professional or Enterprise edition, make sure you leave the VCL Source installation option selected.

6.4.2 XWB_RXE2.bpl File

This run-time package contains the source code for the standard RPC broker components and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation:

C:\Users\Public\Documents\RAD Studio\9.0

6.4.3 XWB_DXE2.bpl File

This design-time package contains the installed components for the standard RPC broker and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation:

C:\Users\Public\Documents\RAD Studio\9.0

6.4.4 SharedRPCBroker_RXE2.bpl File

This run-time package contains the source code for the shared RPC broker components and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation:

C:\Users\Public\Documents\RAD Studio\9.0

6.4.5 SharedRPCBroker_DXE2.bpl File

This design-time package contains the installed components for the shared RPC broker and is found in the following directory after compiling the Broker Development Kit (BDK) on a workstation:

C:\Users\Public\Documents\RAD Studio\9.0

7 RPC Broker Dynamic Link Library (DLL)

7.1 DLL Interface

The RPC Broker provides a Dynamic Link Library (DLL) interface, which acts like a "shell" around the Delphi TRPCBroker component. The DLL is contained in the BAPI32.DLL file.

The DLL interface enables client applications, written in any language that supports access to Microsoft Windows DLL functions, to take advantage of all features of the TRPCBroker component. This allows programming environments other than Embarcadero Delphi to make use of the TRPCBroker component. All of the communication to the server is handled by the TRPCBroker component, accessed via the DLL interface.

7.1.1 Exported Functions

The complete list of functions exported in the DLL is provided in the *RPC Broker Developer's Guide* (i.e., BDK Online Help; Broker_1_1.chm). Functions are provided in the DLL for:

- Creating and destroying RPC Broker components.
- Setting and retrieving RPC Broker component properties.
- Executing RPC Broker component methods.

7.1.2 Header Files Provided

The following header files provide correct declarations for DLL functions:

Table 9. Header files that provide correct declarations for DLL functions

Language	Header File
C	BAPI32.H
C++	BAPI32.HPP
Visual Basic	BAPI32.BAS

7.1.3 Return Values from RPCs

Results from an RPC executed on a VistA M Server are returned as a text stream. This text stream may or may not have embedded **<CR><LF>** character combinations.

When you call an RPC using the TRPCBroker component for Delphi, the text stream returned from an RPC is automatically parsed and returned in the TRPCBroker component's Results property as follows:

Table 10: TRPCBroker component's Results property

Results stream contains <CR><LF> combinations	Location/format of results (assumes RPC's WORD WRAP ON field is True if RPC is Global Array or Word-processing type)
Yes	Results nodes, split based on <CR><LF> delimiter
No	Results[0]

When you call an RPC using the DLL interface, the return value is the unprocessed text stream, which may or may not contain **<CR><LF>** combinations. It is up to you to parse out what would have been individual Results nodes in Delphi, based on the presence of any **<CR><LF>** character combinations in the text stream.

7.1.4 COTS Development and the DLL

The Broker DLL serves as the gateway to the REMOTE PROCEDURE file (#8994) for non-Delphi client/server applications. In order to use any RPCs not written specifically by the client application (e.g., CONSULTS FOR A PATIENT, USER SIGN-ON RPCs, or the more generic VA FileMan RPCs), you must call the RPC Broker DLL with input parameters defined and results accepted in the formats required by the RPC being called.

Therefore, to use the Broker DLL interface you must determine the following information for each RPC you plan to use:

- How does the RPC expect input parameters, if any, to be passed to it?
- Are you able to create any input arrays expected by the RPC in the same format expected by the RPC?
- What does the results data stream returned by the RPC look like?

Glossary

Term	Definition
CLIENT	A single term used interchangeably to refer to the user, the workstation, and the portion of the program that runs on the workstation. In an object-oriented environment, 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).
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.
DHCP	**D**ynamic **H**ost **C**onfiguration **P**rotocol.
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 also carry this code around. With the DLL, you do *not* carry the code itself; you have a pointer to the common library. All applications using it then share one image. 2. DLLs ease maintenance tasks. Because the DLL is a separate file, any modifications made to the DLL do *not* affect the operation of the calling program or any other DLL. 3. DLLs help avoid redundant routines. They provide generic functions that a variety of programs can use.
GUI	**G**raphical **U**ser Interface. 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.
ICON	A picture or symbol that graphically represents an object or a concept.
REMOTE PROCEDURE CALL	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.
SERVER	The computer where the data and the Business Rules reside. It makes resources available to client workstations on the network. In VistA, it is an entry in the OPTION file (#19). An automated mail protocol that is activated by sending a message to a server at another location with the "S.server" syntax. A server's activity is specified in the OPTION file (#19) and can be the running of a routine or the placement of data into a file.

Term	Definition
USER ACCESS	This term is used to refer to a limited level of access to a computer system that is sufficient for using/operating software, but does not allow programming, modification to data dictionaries, or other operations that require programmer/developer access. Any of VistA's options can be locked with a security key (e.g., XUPROGMODE, which means that invoking that option requires programmer/developer 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 software 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.
WINDOW	An object on the screen (dialogue) that presents information such as a document or message.

REF: For a list of commonly used terms and definitions, see the OIT Master Glossary VA Intranet Website.

For a list of commonly used acronyms, see the VA Acronym Lookup Intranet Website.

Index