



RPC BROKER

**GETTING STARTED WITH THE
BROKER DEVELOPMENT KIT (BDK)**

Version 1.1
September 1997

Department of Veterans Affairs
VISTA Software Development
OpenVISTA Product Line

Table of Contents

1. Introduction.....	1-1
2. Orientation	2-1
3. TRPCBroker Component for Delphi.....	3-1
TRPCBroker Component Properties and Methods.....	3-1
TRPCBroker Key Properties.....	3-2
TRPCBroker Key Methods.....	3-3
How to Connect to an M Server.....	3-4
4. Remote Procedure Calls (RPCs)	4-1
What is a Remote Procedure Call?	4-1
Create Your Own RPCs.....	4-1
Writing M Entry Points for RPCs.....	4-2
RPC Entry in the REMOTE PROCEDURE File	4-5
What Makes a Good Remote Procedure Call?.....	4-5
How to Execute an RPC from a Client Application	4-6
RPC Security: How to Register an RPC	4-7
5. Other RPC Broker APIs.....	5-1
GetServerInfo Function	5-1
VISTA Splash Screen Procedures	5-2
XWB GET VARIABLE VALUE RPC	5-3
M Emulation Functions	5-3
Encryption Functions.....	5-4
\$\$BROKER^XWBLIB.....	5-4
\$\$RTRNFMT^XWBLIB.....	5-4
6. Debugging and Troubleshooting.....	6-1
How to Debug Your Client Application.....	6-1
Troubleshooting Connections	6-2
7. RPC Broker Developer Utilities	7-1
Programmer Settings.....	7-1
8. RPC Broker Delphi Package Library (DPL)	8-1
9. RPC Broker Dynamic Link Library (DLL).....	9-1
10. For More Information	10-1
BROKER.HLP Online RPC Broker Developer's Guide	10-1
Other RPC Broker Resources	10-2
Index.....	Index-1

Table of Contents

1. Introduction

The RPC Broker is a client/server system within VA's Veterans Health Information Systems and Technology Architecture (**VISTA**) environment. It enables client applications to communicate and exchange data with M Servers.

This manual introduces developers to the RPC Broker and the Broker Development Kit (BDK). The emphasis is on using the RPC Broker in conjunction with Borland's Delphi software. However, the RPC Broker supports other development environments.

This manual provides an overview of development with the RPC Broker. For more complete information on development with the RPC Broker component, see the *Online RPC Broker Developer's Guide*. It is in Windows Help format, in the file BROKER.HLP, and is distributed in the BDK.

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.

About This Version of the BDK

This version of the BDK provides programmers with the capability to develop new **VISTA** client/server software using the Broker Delphi component (i.e., TRPCBroker) in a 32-bit environment.

To develop **VISTA** applications in a 32-bit environment you must have Delphi V. 2.0 or greater. This version of the RPC Broker component will *not* allow you to develop applications in Delphi V. 1.0. However, the Broker routines on the M server will 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.

Backward Compatibility Issues

Client applications compiled with this version of the RPC Broker (V. 1.1) will not work at a site that has not upgraded its RPC Broker server software to V.1.1.

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

Introduction

2. Orientation

COMMONLY USED TERMS

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

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.  <i>For a more detailed description, see the Borland Delphi for Windows User Guide.</i>
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 reside.

ASSUMPTIONS ABOUT THE READER

This manual is written with the assumption that you have experience working with the following:

- **VISTA** computing environment (e.g., Kernel Installation and Distribution System [KIDS])
- VA FileMan data structures and terminology
- Microsoft Windows
- Borland's Delphi development environment
- GUI standards and guidelines
- M programming language
- Object Pascal programming language

HOW TO USE THIS MANUAL

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

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

- "Snapshots" of computer roll-and-scroll dialogs and computer source code are shown in a non-proportional font. For example:

```
F S I=$O(UNSORTED(I)) Q:I="" S RESULT(UNSORTED(I))=UNSORTED(I)
```

- Object Pascal code uses a combination of upper- and lowercase characters. All Object Pascal reserved words are in boldface type.
- Symbols are used throughout the documentation to alert you to special information, as described in the following table:

Symbol	Description
	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

3. TRPCBroker Component for Delphi

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

TRPCBROKER COMPONENT 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 an M server from your application and execute RPCs on the M server to exchange data and perform actions on the M server.

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

TRPCBROKER KEY PROPERTIES

The following table lists the most important properties of the TRPCBroker component. For a complete list of all of its properties, see the *Online RPC Broker Developer's Guide* (BROKER.HLP), distributed with the BDK.

Property	Description
ClearParameters	If True, the Param property is cleared <i>after</i> every invocation of the Call, strCall, or the lstCall methods.
ClearResults	If True, the Results property is cleared <i>before</i> 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.)



Each of the properties listed above is described in greater detail in the Online RPC Broker Developer's Guide (BROKER.HLP), distributed with the BDK.

TRPCBROKER KEY METHODS

This section lists the most important methods of the TRPCBroker component. For a complete list of all of its methods, see the *Online RPC Broker Developer's Guide* (BROKER.HLP), distributed with the BDK.

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 "How to Execute an RPC from a Client Application" in the Remote Procedure Calls chapter of this manual.

HOW TO CONNECT TO AN M SERVER

To establish a connection from your application to a Broker server:

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:

```

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 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 M server. See the next chapter, Remote Procedure Calls, for information on creating and executing RPCs.

4. Remote Procedure Calls (RPCs)

WHAT IS A REMOTE PROCEDURE CALL?

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

- Send data to an M server
- Execute code on an M server
- Retrieve data from an 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).

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.



CREATE YOUR OWN RPCS

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

WRITING M ENTRY POINTS FOR RPCS

First Input Parameter (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 below) to be returned in this parameter. You must always set some return value into that first parameter before your routine returns.

Return Value Types

There are five RETURN VALUE TYPES for RPCs as shown in the table below. 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.

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, <pre>TAG(RESULT) ; S RESULT="DOE, JOHN" Q</pre>	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, <pre>TAG(RESULT) ; S RESULT(1)="ONE" S RESULT(2)="TWO" Q</pre> 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 setting affects the WORD PROCESSING return value type.	True	Array values, each in a Results item.
		False	Array values, concatenated into Results[0].

RPC Return Value Type	How M Entry Point Should Set First Input Parameter	RPC Word Wrap On Setting	Value(s) returned in Client Results
Global Array	<p>Set the return parameter to a closed global reference in ^TMP. The global's data nodes will be traversed using \$QUERY, and all data values on global nodes descendant from the global reference are returned.</p> <p>This type is especially useful for returning data from VA FileMan word processing fields, where each line is on a 0-subscripted node.</p> <p>Important: 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.</p> <p>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.</p> <p>For example, to return sign-on introductory text you could do:</p> <pre> 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 </pre>	True	Array values, each in a Results item.
		False	Array values, concatenated into Results[0].
Global Instance	<p>Set the return parameter to a closed global reference.</p> <p>For example, to return the 0th node from the NEW PERSON file for the current user:</p> <pre> TAG(RESULT) ; S RESULT=\$NA(^VA(200, DUZ, 0)) Q </pre>	No effect	Value of global node, in Results[0].

Input Parameter Types (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 three format types:

Param PType	Param Value
Literal	Delphi string value, passed as a string literal to the M server.
Reference	Delphi string value, treated on the M Server as an M variable name and resolved from the symbol table at the time the RPC executes.
List	A single-dimensional array of strings in the Mult subproperty of the Param property, passed to the M Server where it is placed in an array. String subscripting can be used.

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.

RPC M Entry Point Examples

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

1. This example takes two numbers and returns their sum:

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

2. This example receives an array of numbers and returns them as a sorted array to the client:

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

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:

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 which 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 distinct array item in Results.
RETURN VALUE TYPE (#.04)	Yes	This indicates to the Broker how to format the return values. For example, if RETURN VALUE TYPE is WORD PROCESSING, then each entry in the returning list will have a <CR><LF> (<carriage return><line feed>) appended.

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)

HOW TO EXECUTE AN RPC FROM A CLIENT APPLICATION

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 subproperties:
 - 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.

Here is an example of some settings of the Param property:

```
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:

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

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.



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.

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:

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



*The OPTION TYPE "B" represents a **Broker** 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) and M code in the RULES subfield that can also determine whether to enable access to each RPC.
3. When you export your package using KIDS, export both your RPCs and your package 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 will be permitted to run.

If the CreateContext method returns False, you should terminate your application (if you don't your application will run, but you will 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 CreateContext to return True.

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.

BrokerExample Online Code Example

The BrokerExample sample application provided with the BDK demonstrates the basic features of developing RPCBroker applications, including:

- Connecting to an M server
- Creating an application context
- Using the GetServerInfo function
- Displaying the **VISTA** splash screen
- Setting the RPCBroker.Param property for each Param Ptype (literal, reference, list)
- Calling RPCs with the Call method
- Calling RPCs with the lstCall and strCall methods

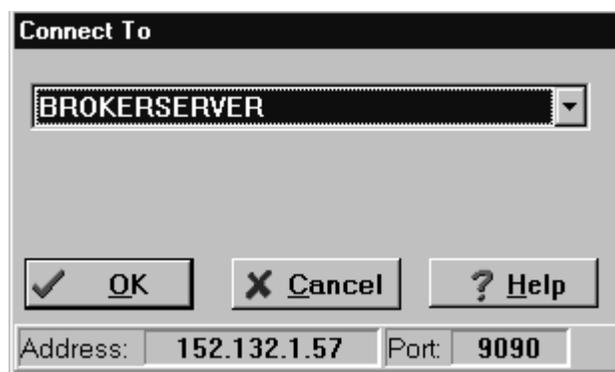
The client source code files for the BrokerExample application are located in the SAMPLES\BROKEREX subdirectory of the main BDK32 directory.

5. Other RPC Broker APIs

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 an application window that allows users to select a service to connect to:



If exactly one server and port entry is defined in the Windows Registry, `GetServerInfo` does not display its dialog window. The values in the single Windows Registry entry are returned, with no user interaction.

If more than one server and port entry exists in the Windows registry, the dialog window is displayed, and the end user can choose which server to connect to.

If no values for server and port are defined in the Windows Registry, `GetServerInfo` does not display its dialog window, and automatic default values are returned (`BROKERSERVER` and `9200`).

Syntax of `GetServerInfo` function:

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



The unit is `RpcConf1`.

VISTA SPLASH SCREEN PROCEDURES

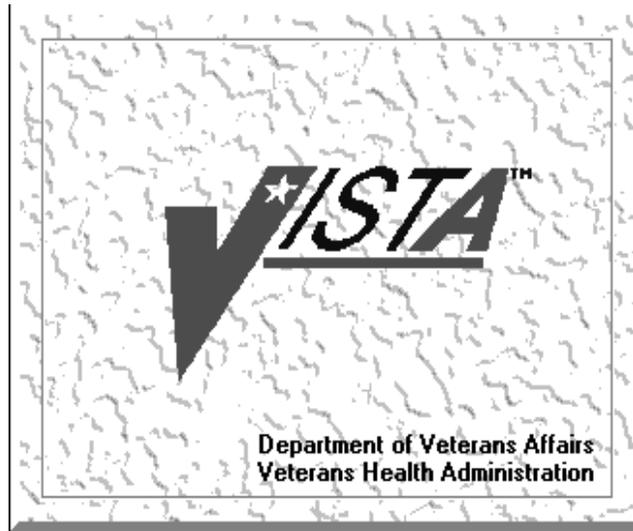
Two procedures in SplVista.PAS unit are provided to display a **VISTA** splash screen when an application loads:

```
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:

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.



```
uses  
  Forms, Unit1 in 'Unit1.pas', SplVista;  
  
{ $R *.RES }  
  
begin  
  Application.Initialize;  
  Application.CreateForm(TForm1, Form1);  
  SplashOpen;  
  SplashClose(2000);  
  Application.Run;  
end.
```

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:

```
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]);
```

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;
```

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;
```

ENCRYPTION FUNCTIONS

Some rudimentary encryption and decryption functions are provided by Kernel and the RPC Broker. Data can be encrypted on the client end and decrypted on the server, and vice-versa.

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;
```

On the M Server

To encrypt:

```
>S CIPHER=$$ENCRYP^XUSRB1("Hello world!") W CIPHER  
/U'11TG~TV1&f-
```

To decrypt:

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

\$\$BROKER^XWBLIB

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

\$\$RTRNFMT^XWBLIB

Use this function in the M code called by an RPC to change the return value type that the RPC will return 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.



For more information about \$\$RTRNFMT^XWBLIB, see the Online RPC Broker Developer's Guide (BROKER.HLP), distributed with the BDK.

6. Debugging and Troubleshooting

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 M server side simultaneously.

To do this:

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 will get a dialog window indicating your workstation IP address and the port number.
2. At this point, switch over to the 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 M server process:

```
>D EN^XWBTCP
```

You will be prompted for the workstation IP address and the port number. After entering the information, switch over to the client application and click on the OK button of the dialog window.

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.

RPC Error Trapping

M errors on the 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 will raise an exception EBrokerError and a popup box displaying the error. At this point RPC execution terminates and the channel is closed.

TROUBLESHOOTING CONNECTIONS

Identifying the Listener Process on the Server

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

Identifying the Handler Process on the Server

On DSM systems the name of a Handler process is `ipXXX.XXX:NNNN`, where `XXX.XXX` are the last two octets of the client IP address and `NNNN` is the port number.

Testing Your RPC Broker Connection

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

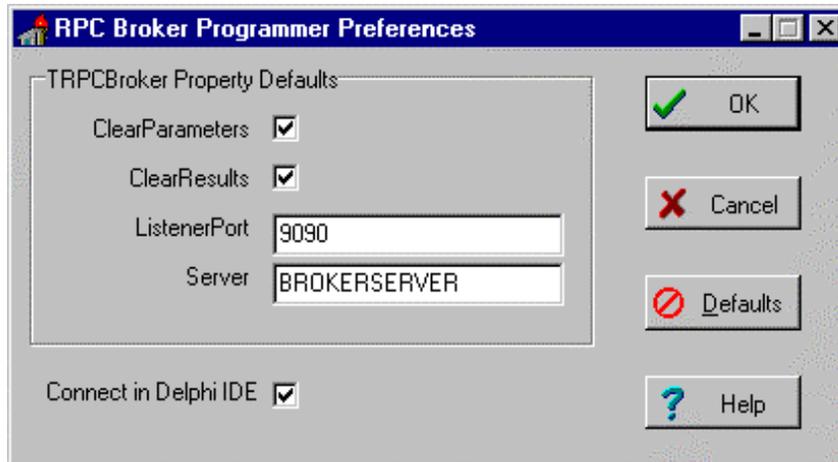


For a complete description of the RPC Broker Diagnostic program, please refer to the Troubleshooting chapter of the RPC Broker Systems Manual.

7. RPC Broker Developer Utilities

PROGRAMMER SETTINGS

You can use BrokerProgPref.exe to define certain default property values for the TRPCBroker component. When you place TRPCBroker component(s) on your form(s) in Delphi, the settings you define are used as the default property values.



You may want to make an entry for BrokerProgPref.exe in Delphi's Tools Menu, to make it easily accessible from within Delphi.

Setting	Description
ClearParameters	If checked, sets the ClearParameters property of a TRPCBroker component to True when you add one to a form.
ClearResults	If checked, sets the ClearResults property to of a TRPCBroker component to True when you add one to a form.
ListenerPort	Sets the ListenerPort property of a TRPCBroker component to the specified value when you add one to a form.
Server	Sets the Server property of a TRPCBroker component to the specified value when you add one to a form.
Connect in Delphi IDE	Enables or disables the connection to the M server from within the Borland Integrated Development Environment (IDE). Disabling this is useful when you are developing in an environment without a connection to an M server. For example, when editing certain server properties, an attempt is made to connect to the Server (if enabled).

8. RPC Broker Delphi Package Library (DPL)

Delphi 3.0 Packages

A new feature with Delphi V. 3.0 enables applications to be distributed in specially compiled dynamic-link libraries called Delphi Package Libraries (DPLs). DPLs enable code-sharing, reduction of executable file size, and conservation of system resources. The use of DPLs is restricted to development in Delphi 3 or higher.

VistaBroker.DPL

You can compile the TRPCBroker component code into your application, or you can choose instead to link your application to the VistaBroker.DPL dynamic link library.



For more information on how to use DPLs, see the Borland Delphi 3 User's Guide.

The VistaBroker.DPL file is installed on end-user workstations by the RPC Broker V. 1.1 end-user client workstation installation program, along with the other RPC Broker V. 1.1. client files. It is installed in an appropriate directory, depending on the operating system, such as \WINDOWS\SYSTEM, such that the DPL is accessible when it is called by application programs. Therefore, you do not need to distribute VistaBroker.DPL with your application.

Distributing the Delphi VCL30.DPL

If you choose to access functionality of the TRPCBroker component through the VistaBroker.DPL dynamic link library, an additional requirement is that the VCL30.DPL library (provided with Delphi V. 3.0) be installed on the end-user workstation.

The RPC Broker does not distribute VCL30.DPL to end-user workstations (it only distributes VistaBroker.DPL). You must ensure that VCL30.DPL is also installed on end-user workstations, perhaps through the installation instructions you provide to system managers.

9. RPC Broker Dynamic Link Library (DLL)

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 file BAPI32.DLL.

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 Borland 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.

Exported Functions

The complete list of functions exported in the DLL is provided in the online Broker Developer's Guide. Functions are provided in the DLL for:

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

Header Files Provided

The following header files provide correct declarations for DLL functions:

C	BAPI32.H
C++	BAPI32.HPP
Visual Basic	BAPI32.BAS

Sample DLL Application

The VBEGCHO sample application, distributed with the Broker Development Kit, demonstrates use of the TRPCBroker DLL from Microsoft Visual Basic.

Return Values from RPCs

Results from an RPC executed on an 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:

Results stream contains <CR><LF> combinations	Location/format of results <i>(assumes RPC's Word Wrap On field is True if RPC is GLOBAL ARRAY or WORD PROCESSING type)</i>
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.

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 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:

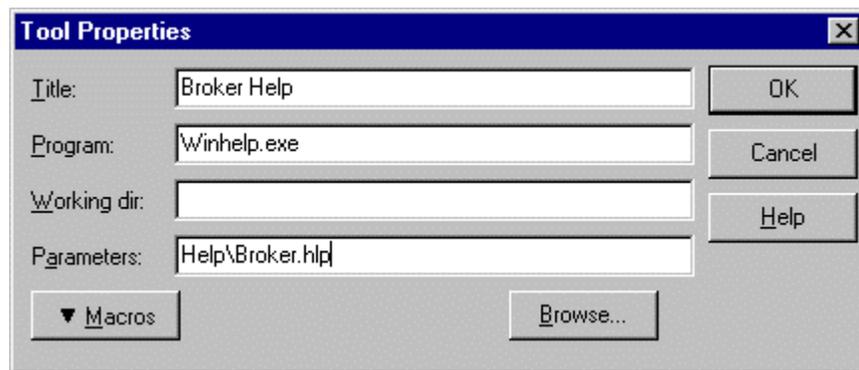
- How does the RPC expect input parameters, if any, to be passed to it?
- Will you be able to create any input arrays expected by the RPC in the same format expected by the RPC?
- What will the results data stream returned by the RPC look like?

10. For More Information

BROKER.HLP Online RPC Broker Developer's Guide

This manual provides an overview of development with the RPC Broker. For more complete information on development with the RPC Broker component, see the *Online RPC Broker Developer's Guide*. It is in Windows Help format, in the file BROKER.HLP, distributed with the BDK.

You may want to make an entry for BROKER.HLP in Delphi's Tools Menu, to make it easily accessible from within Delphi. To do this, use Delphi's Tools | Configure Tools option. Create a new menu entry similar to the following:



The information in the BROKER.HLP file is also available, in HTML format, on the RPC Broker home page:

<http://www.vista.med.va.gov/softserv/infrastr.uct/broker>

BROKER.HLP as Context Sensitive Help Within Delphi

You can install the BROKER.HLP file in such a way that it provides context-sensitive help within Delphi on the TRPCBroker component and its associated properties and methods. This help is available if you have followed the instructions in the *RPC Broker V. 1.1 Installation Guide* to install the Broker help file in Delphi. When installed, you can select the TRPCBroker component or one of its properties in the Object Inspector, and press the F1 key to get help on that item.

For More Information

OTHER RPC BROKER RESOURCES

To learn more about the RPC Broker, consult these related resources:

Title	Format
<i>Online Developer's Guide</i>	Windows Help (BROKER.HLP)
<i>Release Notes</i>	Printed, Acrobat PDF, and HTML
<i>Installation Guide</i>	Printed, Acrobat PDF, and HTML
<i>Systems Manual</i>	Printed and Acrobat PDF
<i>Technical Manual</i>	Printed and Acrobat PDF
<i>Security Guide</i>	Printed and Acrobat PDF

RPC BROKER WEB SITE

Additional information about the RPC Broker, as well as all RPC Broker manuals, are available at the RPC Broker web site:

<http://www.vista.med.va.gov/softserv/infrastr.uct/broker/>

Index

Broker.Hlp	10-1	M Entry Points.....	4-2-4-4
BROKER^XWBLIB.....	5-4	Online Code Samples.....	4-8
BrokerProgPref.exe	7-1	Registering.....	4-7
Bypassing Security (for Development)		RPC Entry	4-5
.....	4-7	REMOTE PROCEDURE file4-1, 4-5,	
Call method.....	3-3, 4-6	9-2	
Context Sensitive Help.....	10-1	RTRNFMT^XWBLIB	5-4
CreateContext method	3-3, 4-7, 5-3	Splash Screen.....	5-2
Debugging	6-1-6-2	SplashClose method	5-2
DECryp^XUSRB1.....	5-4	SplashOpen method.....	5-2
Decrypt method.....	5-4	strCall method	3-3
DPL	8-1	Translate function	5-3
Dynamic Link Library (DLL) ...	9-1-9-2	Troubleshooting.....	6-1-6-2
EN^XWBTCP	6-1	TRPCBroker component.....	3-1-3-4
ENCRYP^XUSRB1	5-4	Call method	3-3
Encrypt method	5-4	Connecting to an M Server.....	3-4
Encryption/decryption functions.....	5-4	CreateContext method...3-3, 4-7, 5-3	
Error Message Handling.....	6-1	lstCall method.....	3-3
GetServerInfo method	3-2, 3-4, 5-1	Methods	3-3
lstCall method.....	3-3	Properties	3-2
Message Handling, Errors.....	6-1	strCall method	3-3
Online Code Samples (RPCs).....	4-8	TRPCBroker Component	
Online Developer's Guide.....	10-1	Call method.....	4-6
OPTION file (#19).....	4-7	Vista Splash Screen.....	5-2
Piece function.....	5-3	VistaBroker.DPL	8-1
Programmer settings.....	7-1	Web site (RPC Broker)	10-2
Registering RPCs.....	4-7	XUPROGMODE security key	4-7
Remote Procedure Calls (RPCs)4-1-4-		XWB GET VARIABLE VALUE	5-3
8		XWBLIB	
Bypassing Security	4-7	\$\$BROKER^XWBLIB	5-4
Executing.....	4-6	\$\$RTRNFMT^XWBLIB.....	5-4

Index