RPC BROKER

GETTING STARTED WITH THE BDK
CHAPTER 3 (Extract): RPC OVERVIEW

Version 1.1

Revised May 2002

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

3. 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:
	`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 setting affects the WORD PROCESSING return value type.	True	Array values, each in a Results item.
		False	Array values, concatenated into Results[0].

<table>
<tr><th>RPC Return Value Type</th><th>How M Entry Point Should Set the Return Parameter</th><th>RPC Word Wrap On Setting</th><th>Value(s) returned in Client Results</th></tr>
<tr><td rowspan="2">Global Array</td><td rowspan="2">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.

This type is especially useful for returning data from VA FileMan word processing fields, where each line is on a 0-subscripted node.

  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 sign-on introductory text you could do:
<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></td><td>True</td><td>Array values, each in a Results item.</td></tr>
<tr><td>False</td><td>Array values, concatenated into Results[0].</td></tr>
<tr><td>Global Instance</td><td>Set the return parameter to a closed global reference.

For example, to return the 0th node from the NEW PERSON file for the current user:
<pre>TAG(RESULT) ;
 S RESULT=$NA(^VA(200,DUZ,0))
 Q</pre></td><td>No effect</td><td>Value of global node, in Results[0].</td></tr>
</table>

Table 4: RPC Broker return value types

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.

Table 5: Input parameter types

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 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 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.

Table 6: REMOTE PROCEDURE file key field entries

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 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.

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.');
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.

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 **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) 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 RPC Broker-based applications, including:

- Connecting to an M server
- Creating an application context
- Using the GetServerInfo function
- Displaying the **V*IST*A** 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.