BASIS Generates SPROC Template Code

Now anyone can write a SPROC!

The SQL language got its start almost forty years ago and is now the standard language for querying, modifying, and managing a relational database. Developers have used SQL for years to manipulate their data, generate reports, and otherwise interact with their BASIS Databases. Despite the power of SQL, however, their data may not be structured for efficient SQL access, having been designed for the fast direct record access offered by the BBx® language syntax. The ability to easily lock data files, extract records, and do keyed reads are more appropriate for these data structures and are second nature to a BBx programmer. The SQL-counterparts are sometimes more complex, difficult to construct, or may not be available in legacy file formats. BASIS recognized this and took steps to aid developers in their quest to write BBx-driven database stored procedures – by providing an easy-to-use code generator that writes customized template code based on a file's data layout.

By Nick Decker
Engineering Supervisor

Stored Procedure Background

Stored procedures, or SPROCs for short, have become popular by helping developers open up their database to third party client access. It is possible to move all of the complex processing logic that previously existed in the primary application into a SPROC. Doing so allows a variety of new clients to access the database while retaining all of the requisite business logic and processing that the application formerly provided. Now that the SPROC is the central location for the business logic, disparate clients can access the database through the SPROC and take advantage of several years' worth of accumulated processing expertise without having to replicate that functionality in every client application.

Another strong case for using SPROCs is their ability to bypass the previous requirement that all SQL access to the database must be on normalized files to avoid an unbearable performance penalty. Many customers were initially pleased with SQL access, only to be disappointed later when queries to non-optimized and non-normalized legacy data files performed poorly. SPROCs in BBj®, with their ability to access record data with traditional READ statements, can use the same speedy routines as the legacy stand-alone applications to retrieve sought-after data, without using SQL or requiring a database overhaul and yet, deliver that data in structures acceptable to third party tools using the SQL language.

SPROCs – a CALL by Another Name

Business BASIC programmers have been using BBx's CALL statement for decades to access commonly used code and libraries of routines. In order to interface with the CALLed program effectively, a BBx program passes variables along in the CALL statement and the CALLed program uses an ENTER statement to retrieve these values. The following line of code should look familiar to most BBx programmers:

```
CALL "CUSTINFO", CUSTNUM, COMPANY$
```

In the same way that the CALL verb allows BBx programs to invoke another program, passing in variables and receiving information back, a SPROC does the same for client access to a database. In fact, even though we are now using SQL to talk to the database, the syntax is remarkably similar:

```
CALL CUSTINFO(CUSTNUM,'COMPANY')
```

>>

Not only does the syntax look strikingly similar, but developers use SPROCs in large part for the same reasons – launching a program that provides a commonly used service for several applications. You can also pass data into a SPROC in much the same way that you specify variables to pass to a public program. SPROCs are very flexible as they provide the traditional input and output variables along with advanced return types such as return codes and full-blown result sets, similar to a grid full of data.

Creating a new SPROC

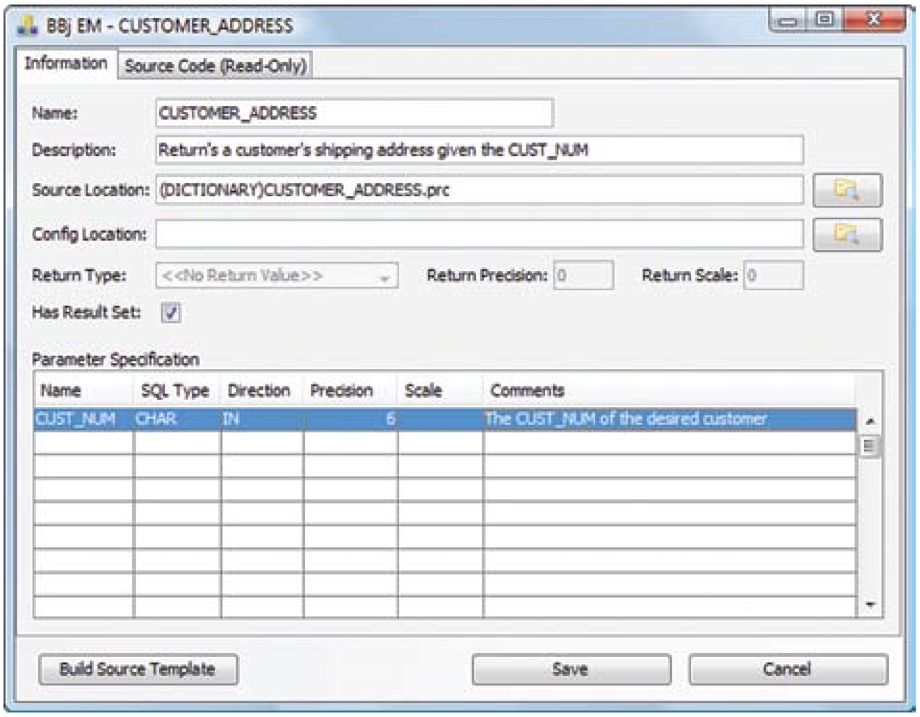
So now that everyone is sold on SPROCs and are reassured that the code is familiar, how does one go about creating one of these "called" routines/programs?

The first step is to determine what sort of database functionality the SPROC will provide, followed by the desired input and output data. In our example, we use the ChileCompany demo database and create a SPROC that returns a customer's billing/shipping address given their customer number.

Next, launch Enterprise Manager (EM) or load the EM module from within the BASIS IDE for a truly all-in-one experience. Then select the ChileCompany database entry from the list of databases in the bottom left panel. Click on the Procedures tab in the right-hand information pane for the ChileCompany, then the Plus button to add a new SPROC. A new window, as shown in **Figure 1**, appears in which you can enter all of the necessary information and full description of the new SPROC.

In this example, the SPROC's name is CUSTOMER_ADDRESS. The program file **`CUSTOMER_ADDRESS.prc`**, located in the Data Dictionary directory, runs when a client calls the SPROC. Selecting the "Has Result Set" checkbox indicates that it returns a result set back to the caller. The result set is a handy way to return data back to the client, especially if the SPROC might return multiple rows of data. In our example, the SPROC only returns a single row – the address for the specified customer – but the result set makes it easy to return several rows and columns worth of data.

Lastly, we have defined a single parameter called CUST_NUM that is a CHAR type with a direction of IN. This means that the client must provide a string containing the customer number when calling the SPROC to specify which customer address they require.



BBj EM - CUSTOMER_ADDRESS

Information | Source Code (Read-Only)

Name: CUSTOMER_ADDRESS
Description: Return's a customer's shipping address given the CUST_NUM
Source Location: (DICTIONARY)CUSTOMER_ADDRESS.prc
Config Location:
Return Type: <<No Return Value>> Return Precision: 0 Return Scale: 0
Has Result Set: ✓

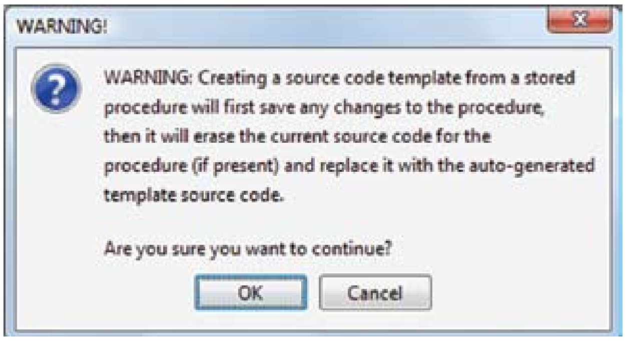
Parameter Specification

Name	SQL Type	Direction	Precision	Scale	Comments
CUST_NUM	CHAR	IN	6		The CUST_NUM of the desired customer

Figure 1. Adding a new stored procedure

EM Generates SPROC Code Template

The next step is where the magic happens. By clicking the [Build Source Template] button, the Enterprise Manager writes most of the SPROC code needed to make the program viable. Clicking the button results in the dialog box shown in **Figure 2** that warns it will overwrite the designated program file **`(DICTIONARY)CUSTOMER_ADDRESS.prc`** with a newly-generated template. If we had previously written the code for the SPROC and pushed the button by mistake, selecting [Cancel] would abort the process. Since we have not written the SPROC program, selecting [OK] causes EM to write the SPROC program file.



WARNING!

WARNING: Creating a source code template from a stored procedure will first save any changes to the procedure, then it will erase the current source code for the procedure (if present) and replace it with the auto-generated template source code.

Are you sure you want to continue?

Figure 2. EM's warning when creating a new template SPROC program

Customizing the SPROC's Output

Enterprise Manager gives us the opportunity to specify a string template to describe the SPROC's return result set. When the generator first created the SPROC, it only specified that the new procedure would return a result set of data back to the client. At that point in time, it was not critical to define what that result set would look like – only that the SPROC would use it to return data to the client. This makes sense, as a SPROC can be very flexible, returning different result sets back depending on which type of input parameters the client supplied. But now that the developer is getting down to the nitty-gritty of the SPROC code itself, it is time to figure out exactly what the return result set should look like.

For our example, the invoker of the SPROC needs the shipping address for the customer, so the data will be a subset of the appropriate record in the ChileCompany's CUSTOMER data

>>

file. To access the CUSTOMER table's string template, click on the Tables tab in EM then double-click the CUSTOMER entry in the table list. The resultant window brings up the properties for the CUSTOMER table and offers a button called [String Template] as shown in **Figure 3**. Click the button to show the string template for the data file in a text box and copy the address portion into the clipboard.

Equipped with the desired section of the customer table's string template, paste it into EM's dialog to define the columns that comprise a record in the return result set (see **Figure 4)**.

After clicking [OK], EM writes out the template SPROC code. Now save the fully defined SPROC to test it. Notice that the list of stored procedures in EM now contains the new SPROC at the top of the list as shown in **Figure 5**.

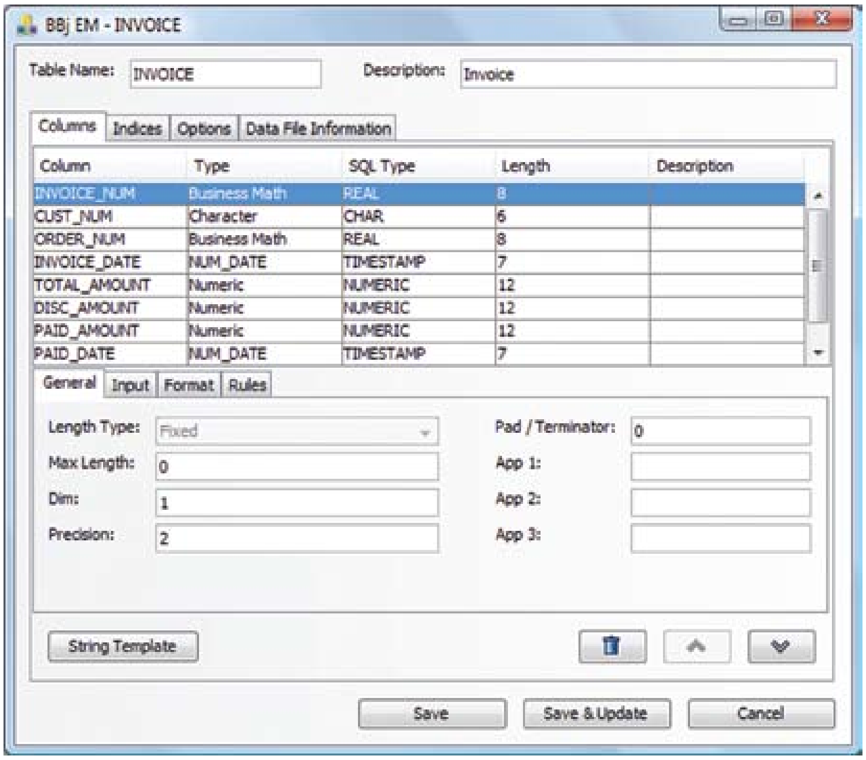

BBj EM - INVOICE

Table Name: INVOICE Description: Invoice

Columns | Indices | Options | Data File Information

Column	Type	SQL Type	Length	Description
INVOICE_NUM	Business Math	REAL	8	
CUST_NUM	Character	CHAR	6	
ORDER_NUM	Business Math	REAL	8	
INVOICE_DATE	NUM_DATE	TIMESTAMP	7	
TOTAL_AMOUNT	Numeric	NUMERIC	12	
DISC_AMOUNT	Numeric	NUMERIC	12	
PAID_AMOUNT	Numeric	NUMERIC	12	
PAID_DATE	NUM_DATE	TIMESTAMP	7	

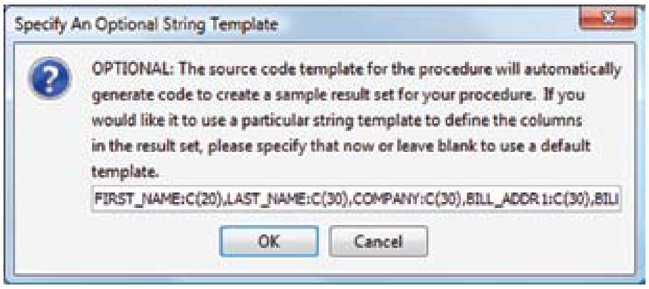
General | Input | Format | Rules

Length Type:	Fixed	Pad / Terminator:	0
Max Length:	0	App 1:	
Dim:	1	App 2:	
Precision:	2	App 3:	

String Template

Save | Save & Update | Cancel

Figure 3. Accessing a data file's string template


Specify An Optional String Template

OPTIONAL: The source code template for the procedure will automatically generate code to create a sample result set for your procedure. If you would like it to use a particular string template to define the columns in the result set, please specify that now or leave blank to use a default template.

FIRST_NAME:C(20),LAST_NAME:C(30),COMPANY:C(30),BILL_ADDR1:C(30),BILL

OK | Cancel

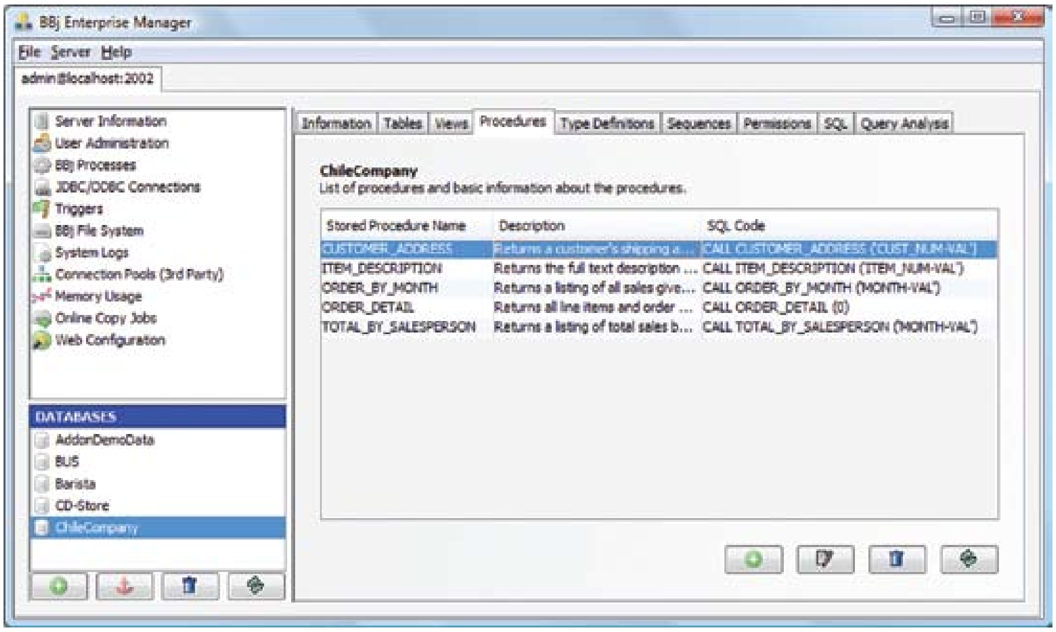
Figure 4. Specifying the string template for the result set

Testing the SPROC

Because EM wrote a full-blown program for the backend of the SPROC, it is possible to try it out right away. Obviously, it will not work exactly as needed, since EM cannot read minds and the developer did not tell it from which file to get the data. However, it did create a fully functioning program that will return sample data when CALLed by a client. To take it out for a test drive, execute an SQL CALL statement in the SQL tab. EM saves time here again, as the SPROC listing (as shown previously in **Figure 5**) also contains sample SQL code to invoke the SPROC. The simplest way to proceed is to copy the sample SQL code from the SPROC line, click over to the SQL tab, and then paste it into the SQL Statement box (see **Figure 6**).

Executing the SQL statement causes the BASIS DBMS to run the BBx program that defines the SPROC, returning a result set with the fields specified in the string template. After verifying that the new SPROC works without error, a fully functional SPROC is just around the corner, having already created a new SPROC, asked EM to write out a functional template program, and ensured that the SPROC actually works. The final step is

>>


BBj Enterprise Manager

File Server Help

admin@localhost:2002

Server Information
User Administration
BBj Processes
JDBC/ODBC Connections
Triggers
BBj File System
System Logs
Connection Pools (3rd Party)
Memory Usage
Online Copy Jobs
Web Configuration

DATABASES
AddonDemoData
BUS
Barista
CD-Store
ChileCompany

Information | Tables | Views | Procedures | Type Definitions | Sequences | Permissions | SQL | Query Analysis

ChileCompany
List of procedures and basic information about the procedures.

Stored Procedure Name	Description	SQL Code
CUSTOMER_ADDRESS	Returns a customer's shipping a...	CALL CUSTOMER_ADDRESS ('CUST_NUM-VAL')
ITEM_DESCRIPTION	Returns the full text description ...	CALL ITEM_DESCRIPTION ('ITEM_NUM-VAL')
ORDER_BY_MONTH	Returns a listing of all sales give...	CALL ORDER_BY_MONTH ('MONTH-VAL')
ORDER_DETAIL	Returns all line items and order ...	CALL ORDER_DETAIL (0)
TOTAL_BY_SALESPERSON	Returns a listing of total sales b...	CALL TOTAL_BY_SALESPERSON ('MONTH-VAL')

Figure 5. The newly-defined SPROC in the list for the ChileCompany

to modify the EM-generated template program to return the right data – the actual data from the ChileCompany CUSTOMER data file.

Modifying the SPROC Program

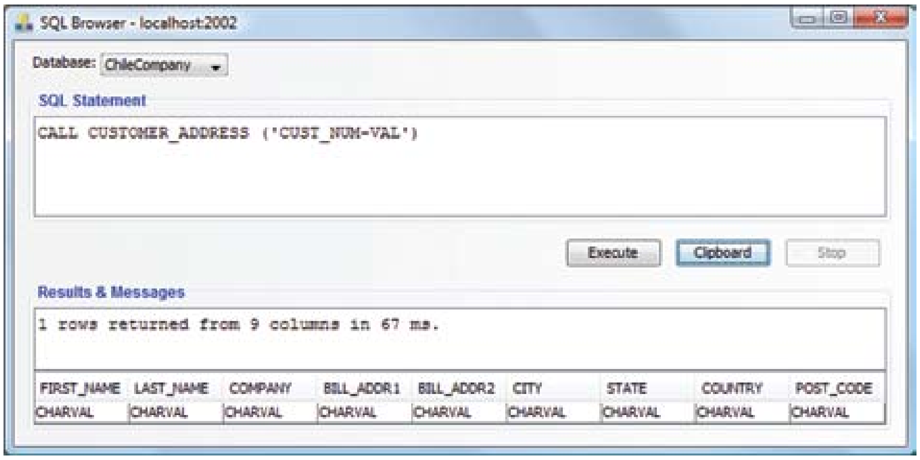
Open up the **`CUSTOMER_ADDRESS.prc`** file in the BASIS IDE and notice that the EM-generated program file already does most of the work. For example, the program takes care of getting the customer number that the client specified and loads it into a variable named CUST_NUM$. It also creates a memory record set, fills it with the appropriate address data, and sends it back to the client. The only change needed, is to remove the section of code that fills the record set with sample data and replace it with code that fills it with the real data. That job is easy too, as EM already wrote most of that code based on the supplied string template. In fact, just make a couple of small changes and the SPROC will retrieve data from the designated CUSTOMER data file.

Begin by removing the section of code that fills the record set with sample data, as shown in **Figure 7**.

Notice that the code filled every field in the record set with the string CHARVAL. This should look familiar as it is the same return result set for every column that occurred on the first test of the SPROC from the EM.

The next step is to enable the template code that fills the record set with the correct data from the CUSTOMER data file. After uncommenting the pre-generated code block, it looks like **Figure 8**.

The code is very close to what is ultimately desired, but it still needs a few changes. For starters, specify the path to the actual CUSTOMER data file instead of the MY_FILE placeholder in the code. Next, modify the rec$ template to match the full string template for the CUSTOMER file (right now it matches the return result set template which is just a subset of the full record template). Lastly, modify the READ RECORD routine. The routine is designed to read all the way through a data file, returning every record. The goal with the customer address is different, >>



```
CALL CUSTOMER_ADDRESS ('CUST_NUM-VAL')
```

```
1 rows returned from 9 columns in 67 ms.
```

FIRST_NAME	LAST_NAME	COMPANY	BILL_ADDR1	BILL_ADDR2	CITY	STATE	COUNTRY	POST_CODE
CHARVAL	CHARVAL	CHARVAL	CHARVAL	CHARVAL	CHARVAL	CHARVAL	CHARVAL	CHARVAL

Figure 6. Testing the newly created SPROC

```
REM Populate the record set with some sample data.
REM If you uncomment the code above, you will want to
REM make sure to remove this code.
data! = rs!.getEmptyRecordData()
data!.setFieldValue("FIRST_NAME", "CHARVAL")
data!.setFieldValue("LAST_NAME", "CHARVAL")
data!.setFieldValue("COMPANY", "CHARVAL")
data!.setFieldValue("BILL_ADDR1", "CHARVAL")
data!.setFieldValue("BILL_ADDR2", "CHARVAL")
data!.setFieldValue("CITY", "CHARVAL")
data!.setFieldValue("STATE", "CHARVAL")
data!.setFieldValue("COUNTRY", "CHARVAL")
data!.setFieldValue("POST_CODE", "CHARVAL")
rs!.insert(data!)
```

Figure 7. Filling the record set with sample data

```
REM ***************************************************************
REM This portion is commented out.  Make necessary changes
REM to this section to make your procedure read records from
REM a BBj data file and populate the in-memory recordset
REM with the results.
REM
REM Create a template to hold the read record data.
DIM rec$:"FIRST_NAME:C(20),LAST_NAME:C(30),COMPANY:C(30),BILL_ADDR
dataFile$ = "../data/MY_FILE"
    chan = unt
    open(chan) dataFile$
    while 1
        read record(chan,err=*BREAK) rec$
        data! = rs!.getEmptyRecordData()
        data!.setFieldValue("FIRST_NAME", rec.FIRST_NAME$)
        data!.setFieldValue("LAST_NAME", rec.LAST_NAME$)
        data!.setFieldValue("COMPANY", rec.COMPANY$)
        data!.setFieldValue("BILL_ADDR1", rec.BILL_ADDR1$)
        data!.setFieldValue("BILL_ADDR2", rec.BILL_ADDR2$)
        data!.setFieldValue("CITY", rec.CITY$)
        data!.setFieldValue("STATE", rec.STATE$)
        data!.setFieldValue("COUNTRY", rec.COUNTRY$)
        data!.setFieldValue("POST_CODE", rec.POST_CODE$)
        rs!.insert(data!)
    wend
```

Figure 8. The template code that fills the record set

though, because it just needs to return a single address from the customer file. Since the SPROC has an input parameter for CUST_NUM, the caller can specify which customer address to return. Therefore, remove the WHILE/WEND loop and add a KEY= mode to the READ RECORD statement so that the code only reads the record the client requests. The resulting code appears in **Figure 9**.

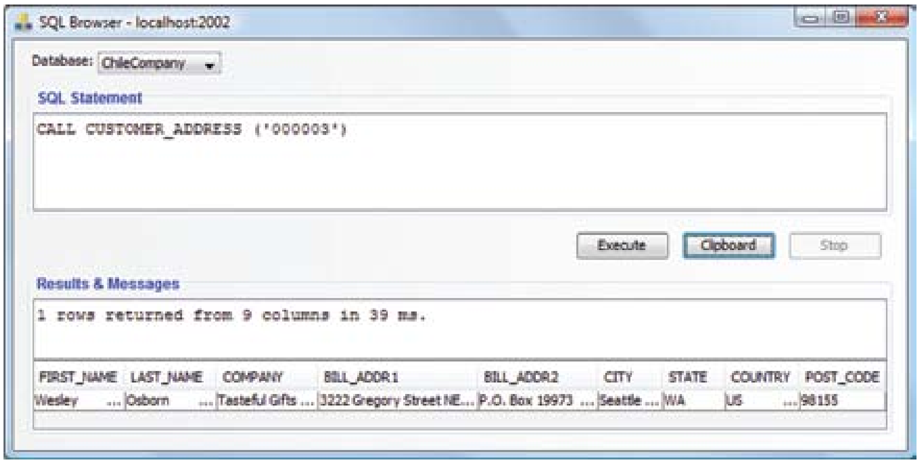
Testing the Completed SPROC

With the final code changes in place, it is time to test the SPROC once again. This time supply an actual customer number for the input parameter and the SPROC will return live data from the ChileCompany Customer data file. In EM's SQL tab, run the query once again but this time specify a six-digit customer number as the input parameter. The SPROC will successfully retrieve the customer's shipping address from the database and display it as a result set in a grid as shown in **Figure 10.**

```
REM ********************************************************
REM This portion is commented out.  Make necessary changes
REM to this section to make your procedure read records from
REM a BBj data file and populate the in-memory recordset
REM with the results.
REM Create a template to hold the read record data.
DIM rec$:"CUST_NUM:C(6),FIRST_NAME:C(20),LAST_NAME:C(30),COMPANY
dataFile$ = "../data/CUSTOMER"
    chan = unt
    open(chan) dataFile$
    read record(chan,key=CUST_NUM$,err=*BREAK) rec$
    data! = rs!.getEmptyRecordData()
    data!.setFieldValue("FIRST_NAME", rec.FIRST_NAME$)
    data!.setFieldValue("LAST_NAME", rec.LAST_NAME$)
    data!.setFieldValue("COMPANY", rec.COMPANY$)
    data!.setFieldValue("BILL_ADDR1", rec.BILL_ADDR1$)
    data!.setFieldValue("BILL_ADDR2", rec.BILL_ADDR2$)
    data!.setFieldValue("CITY", rec.CITY$)
    data!.setFieldValue("STATE", rec.STATE$)
    data!.setFieldValue("COUNTRY", rec.COUNTRY$)
    data!.setFieldValue("POST_CODE", rec.POST_CODE$)
    rs!.insert(data!)

REM Tell the stored procedure to return the result set.
sp!.setRecordSet(rs!)
```

Figure 9. The final READ RECORD code in the SPROC



Database: ChileCompany

```
CALL CUSTOMER_ADDRESS ('000003')
```

```
1 rows returned from 9 columns in 39 ms.
```

FIRST_NAME	LAST_NAME	COMPANY	BILL_ADDR1	BILL_ADDR2	CITY	STATE	COUNTRY	POST_CODE
Wesley ...	Osborn ...	Tasteful Gifts ...	3222 Gregory Street NE...	P.O. Box 19973 ...	Seattle ...	WA	US ...	98155

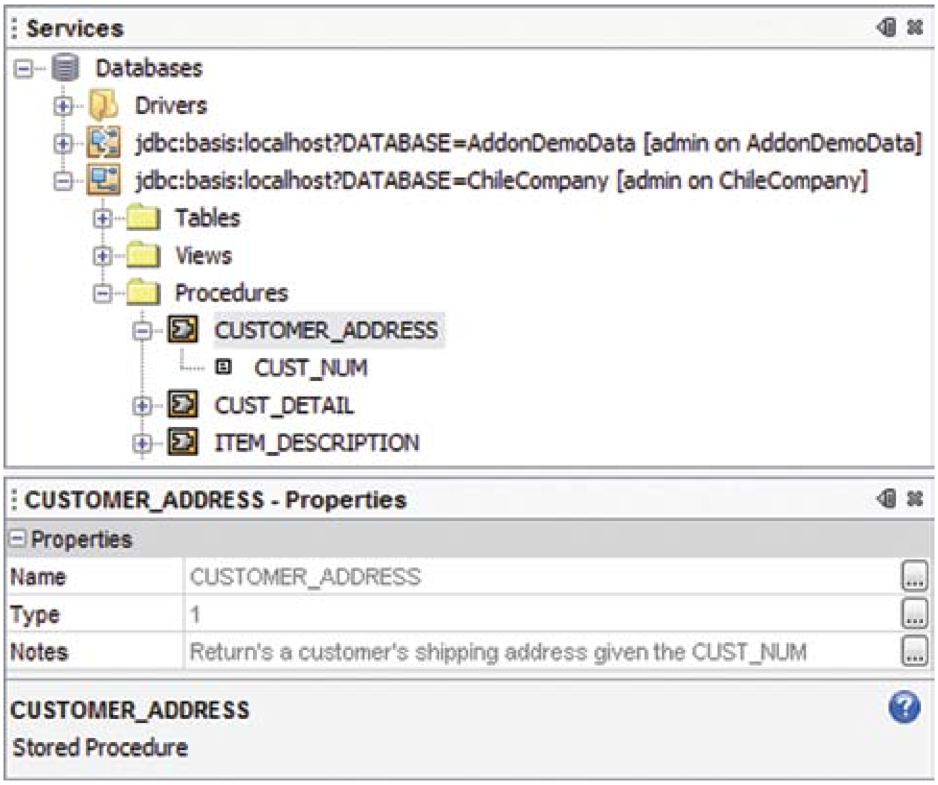
Figure 10. Testing the SPROC in EM

With a fully functional SPROC, a plethora of applications can access the customer data via SQL. The iReport Designer, covered in more detail in this issue's *Recipes for Successful Report Writing* on page 6, is a perfect example. Report authors traditionally use SQL queries to retrieve data for their reports, and a call to the new SPROC is just the ticket. After all, the report data can be the result of any query – whether it is accessing a table, view, or SPROC. In **Figure 11**, iReport's Services section of the IDE demonstrates this by dutifully offering a list of available tables, views, and SPROCs for each database connection.

The new SPROC, CUSTOMER_ADDRESS, shows up in the list of available procedures for the ChileCompany Database. Expanding the SPROC node reveals the list of its parameters. For each database connection, iReport talks to the back-end database and gathers metadata regarding the tables, views, and stored procedures. iReport takes this a step further by gathering details such as the description for each SPROC and the data types and comments for every parameter. The screenshot in **Figure 11** demonstrates this by displaying the input parameter and description for this new SPROC in its properties window. Notice that the Notes field in the properties window reflects the SPROC description that was filled in when originally creating the SPROC (see **Figure 1**). By including helpful comments and descriptions for SPROCs and their parameters, it makes it much easier for the end users to create a report because the SPROCs are self-explanatory. When report authors load up a tool like iReport, they have access to the right data without needing intimate knowledge of the database. All they have to do is peruse a list of available SPROCs and read the descriptions of each to find out which one suits their purpose. Supplied with the built-in descriptions and comments for each parameter, they are well on their way to creating a report in a flash. **>>**

Summary

Anyone who followed these steps has completed their first stored procedure and it is ready for action. Not bad for a few minutes worth of work! Armed with Enterprise Manager's new SPROC template generation and the fact that BBj stored procedures can leverage existing legacy code and utilize standard BBx syntax like READ RECORDs, creating SPROCs are now a snap. Legacy programmers are empowered to expose their databases and business logic to other SQL applications safely, securely, and with the blazing speed of native access. ■



```
Services
Databases
    Drivers
    jdbc:basis:localhost?DATABASE=AddonDemoData [admin on AddonDemoData]
    jdbc:basis:localhost?DATABASE=ChileCompany [admin on ChileCompany]
        Tables
        Views
        Procedures
            CUSTOMER_ADDRESS
                CUST_NUM
            CUST_DETAIL
            ITEM_DESCRIPTION
```

CUSTOMER_ADDRESS - Properties

Properties

Name	CUSTOMER_ADDRESS
Type	1
Notes	Return's a customer's shipping address given the CUST_NUM

CUSTOMER_ADDRESS
Stored Procedure

Figure 11. iReport shows the available SPROCs, parameters, and descriptions