Why Use the BBjRecordSet?

By Jim Douglas

In the 4th Quarter 2003 issue of the *BASIS International Advantage*, we described some specific uses for the BBjRecordSet object in the how-to articles, "Using the BBjRecordSet" and "BBj Databound Controls." In this follow-up article, we will step back a bit from the low-level details to talk about some reasons to use the BBjRecordSet in an application.

What Is A BBjRecordSet?

A BBjRecordSet is a logical view of a set of records defined in terms of an SQL SELECT or a BBj® multi-keyed file. BBjRecordSets are useful in several ways:

- Unlike a standard BBj SQL SELECT channel, an SQL-based BBjRecordSet is directly updatable using the `insert()`, `update()` and `deleteCurrentRecordData()` methods
- The developer can navigate freely back and forth through a BBjRecordSet using `first()`, `next()`, `previous()`, `last()` and `move(+n/-n)`
- Using the `seek()`, `seekForward()`, and `seekBackward()` methods, the developer can scan through a BBjRecordSet for records that match any criteria
- BBjRecordSets provide the underlying plumbing to support Databound Controls

The following example uses a BBjRecordSet to print a list of customers:

```
db$="ChileCompany",query$="select * from customer order by last_name"
rs! = BBjAPI().createSQLRecordSet(db$,"",query$)
rs!.first()
while 1
   rd!=rs!.getCurrentRecordData()
   print rd!.getFieldValue("cust_num")," ",
   print cvs(rd!.getFieldValue("last_name"),3),", ",
   print cvs(rd!.getFieldValue("first_name"),3)
   rs!.next(err=*break)
wend
```

The example above shows that the `getCurrentRecordData()` method retrieves the current record of a BBjRecordSet. The navigation methods (`first()`, `next()`, `last()`, `previous()`, `move()`) and seek methods (`seek()`, `seekForward()`, `seekBackward()`) all set the current record.

It would be just as easy to print that list backwards by using `last()` instead of `first()` and `previous()` instead of `next()`:

```
db$="ChileCompany",query$="select * from customer order by last_name"
rs! = BBjAPI().createSQLRecordSet(db$,"",query$)
rs!.last()
while 1
   rd!=rs!.getCurrentRecordData()
   print rd!.getFieldValue("cust_num")," ",
   print cvs(rd!.getFieldValue("last_name"),3),", ",
   print cvs(rd!.getFieldValue("first_name"),3)
   rs!.previous(err=*break)
wend
```

BBjRecordSets are either editable (read/write) or non-editable (read-only). A read/write BBjRecordSet has the ability to insert, update, or delete records from the BBjRecordSet. The underlying file or database automatically reflects all changes made to a read/write BBjRecordSet. File-based BBjRecordSets are usually read/write as long as the underlying file is read/write and the query includes the primary key. SQL-based BBjRecordSets are read-only if the query doesn't include the primary key or the user made enough modifications to the returned fields (functions, concatenations or truncations, etc.) to prevent the system from tracing the record back to the underlying data.

continued...

Searching and Seeking

One way to visualize a BBjRecordSet is as a table containing numbered BBjRecordData objects starting at zero. For example, the following query produced the table in **Figure 1**.

```
db$="ChileCompany"
query$="select cust_num,last_name,first_name from customer order by
last_name"
rs! = BBjAPI().createSQLRecordSet(db$,"",query$)
```

(#)	cust_num	last_name	first_name
0	000029	Abbott	Meghan
1	000032	Adams	Barbara
2	000015	Alcott	Amy
3	000013	Augustine	Ernie
4	000063	Aurora	Justin
5	000069	Bakkund	Bjorn
6	000001	Baldrake	Gregory
7	000007	Booker	Dr. Jane
8	000049	Bowerman	Bob
9	000041	Calloway	Tony
10	000048	Caroll	Stuart
58	000017	Tomlin	Myrah
59	000037	Tookeypatch	Claudia
60	000047	Troy	Darlene
61	000026	Tuscany	John
62	000065	Van Dyke	Michael
63	000021	Wellesley	Dianne
64	000031	Wesson	Lawrence
65	000025	Williams	Doris
66	000044	Wittenberg	Roberta
67	000068	Zamora	Carlos

Figure 1. The BBjRecordSet resulting from the code sample

The developer can scan through this BBjRecordSet in various ways such as seeking to the first, next, previous or last record, or moving backward or forward a specific number of records. For example:

```
>rs!.first()
>?rs!.getCurrentRecordData()
last_name=Abbott
first_name=Meghan
cust_num=000029

>rs!.next()
>?rs!.getCurrentRecordData()
last_name=Adams
first_name=Barbara
cust_num=000032

>rs!.move(+5)
>?rs!.getCurrentRecordData()
last_name=Baldrake
first_name=Gregory
cust_num=000001
```

continued...

In addition to the previous examples, the next example shows how to seek for records containing specific field information such as all customers named "John."

```
>seek!=rs!.getEmptyRecordData()
>seek!.setFieldValue("first_name","John")
>rs!.seek(seek!,1)
>?rs!.getCurrentRecordData()
last_name=Roberts
first_name=John
cust_num=000016

>rs!.seekForward(seek!,1)
>?rs!.getCurrentRecordData()
last_name=Tuscany
first_name=John
cust_num=000026

>rs!.seekForward(seek!,1)
!ERROR=80  (No Matching Entry Found)
>
```

The next example prints a list of customers with no purchases this year by using a filter on the SQL select:

```
db$="ChileCompany",query$="select * from customer where sales_ytd=0 order by last_name"
rs! = BBjAPI().createSQLRecordSet(db$,"",query$)
rs!.first()
while 1
   rd!=rs!.getCurrentRecordData()
   print rd!.getFieldValue("cust_num”)," ",
   print cvs(rd!.getFieldValue("last_name"),3),", ",
   print cvs(rd!.getFieldValue("first_name"),3)
   rs!.next(err=*break)
wend
```

The following example shows a different way to retrieve the same set of records (customers with no purchases this year):

```
db$="ChileCompany"
query$="select * from customer order by last_name"
rs! = BBjAPI().createSQLRecordSet(db$,"",query$)
Filter! = rs!.getEmptyRecordData()
Filter!.setFieldValue("sales_ytd","0")
rs!.seek(Filter!)
while 1
   rd!=rs!.getCurrentRecordData()
   print rd!.getFieldValue("cust_num")," ",
   print cvs(rd!.getFieldValue("last_name"),3),", ",
   print cvs(rd!.getFieldValue("first_name"),3)
   rs!.seekForward(Filter!,err=*break)
wend
```

Filtering records via the SQL SELECT statement is generally the preferred method, as the resultant record set will be much smaller. Smaller record sets translate into less data transferred from the server to the client, which is desirable in three-tier

continued...

deployments. Seeks for specific data in the recordset occur faster as well, since the `seekForward()` and `seekBackward()` methods always scan every record to the end of the recordset, even in seeks that only involve key fields. Though the previous two examples return the same information, the first example executes in half the time because the resultant record set contains only 28 records compared to 68 records in the second set.

Locking

BBjRecordSets do not allow for record locking. If there is any chance that another process might access the file(s) referenced in a BBjRecordSet at the same time, then all attempts to navigate through or modify the BBjRecordSet must include an ERR= option to handle the case when the underlying record is locked or otherwise unavailable.

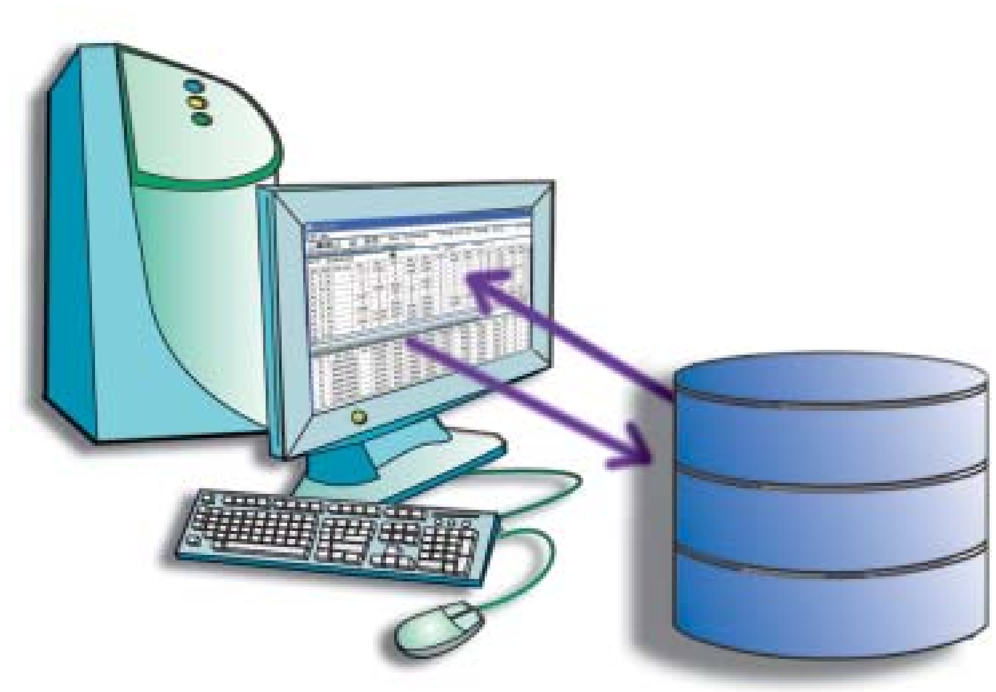
Databound Controls

BBj Databound controls use BBjRecordSets and have several benefits:

- Defining the link between the GUI control and the associated data field on disk in a single location (either a resource file or at the beginning of a program).
- Associating list controls (BBjListBox, BBjListButton, BBjListEdit) with a recordset to automatically populate them with data.
- Limiting application procedural code to validating data, deciding when to insert, update, or delete records, and moving the record pointer. With the addition of a BBjNavigator control, much of the navigation becomes automatic.

BBj GUI Control	**Binds To Field of Type:**	**Affect on GUI control** *when current record data changes*:	**Affect on current record data** *when the GUI control is modified:*
BBjStaticText	String	The **Static Text** control displays the string field from the record data.	N/A (The user cannot change **Static Text**.)
BBjEditBox	String	The **Edit Box** displays the string field from the record data.	The record data automatically reflects any changes in the **Edit Box**.
BBjCheckBox	Number	The **Check Box** shows a check if the numeric field in the record data has a non-zero value.	The record data stores the number one for a checked checkbox or a zero for an unchecked **Check Box**.
BBjCEdit	String	The **Custom Edit** displays the string field from the record data.	The record data automatically reflects any changes in the **Custom Edit**.
BBjListBox	String	The **List Box** selects the index that corresponds to the string field from the record data.	The record data automatically reflects the string value of the selected index.
BBjListButton	String	The **List Button** selects the index that corresponds to the string field from the record data.	The record data automatically reflects the string value of the selected index.
BBjListEdit	String	The **Edit Box** of the BBjListEdit control displays the string field from the record data.	The record data automatically reflects any changes to the **Edit Box**, or selections from the list.
BBjInputE	String	The **InputE** control displays the string field from the record data.	The record data automatically reflects any changes in the **InputE** control.
BBjInputN	Number	The **InputN** control displays the numeric field from the record data.	The record data automatically reflects any changes in the **InputN** control.
BBjInputD	Number	See the rules for binding date fields to grid cells in BBj Data Aware Grid Channels	
BBjNavigator	String	The **Navigator** label displays the string field from the record data.	N/A (The user cannot change the **Navigator** label.)

continued...



The previous table summarizes all of the BBj GUI controls that a programmer may bind to a BBjRecordSet field:

So, Why Use BBjRecordSet?

BBjRecordSets offer developers a new and powerful way to view and manipulate data. Not only do they provide extensive capabilities in an easy-to-understand object-oriented manner, but they abstract the data, freeing the programmer from the detailed low-level file I/O operations that were necessary in the past. Using BBjRecordSets, developers can quickly access selected data in a few lines of code, and then easily search and manipulate that subset. After using BBjRecordSets a couple of times, developers may wonder how they ever programmed without them!

For more information about BBjRecordSets and Databound Controls, refer to the **BBjRecordSet** API documentation and the articles "Using the BBjRecordSet" and "BBj Databound Controls" in the 4th Quarter 2003 issue of the *BASIS International Advantage*.