BASIS Adds Mapping Functionality to BBjRecordSets

“Suddenly, there came a RecordSet Mapping, Mapping at my Chamber Door!”

By Jon Bradley

Databound BBj controls make it easy to display and modify data in a GUI application. Using AppBuilder’s FormGen Wizard, it is possible to create an entire file maintenance application without writing a single line of code. New in BBj 8.0, mapped recordsets make it possible to customize and enhance such an application easily.

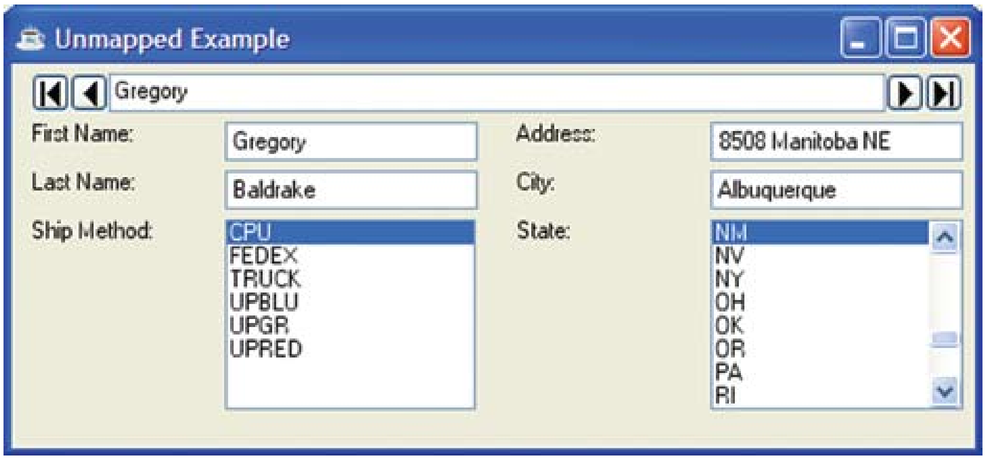
Trouble can rear its ugly head when displaying stored data. Typically, this data is condensed for storage purposes and therefore not in a format or expression familiar to the user. BBj 8.0 addresses this issue by introducing the ability to add display translations or "mappings" to your BBjRecordSets.

For example, Julian dates are stored in data files in a numerical format such as `2454628`. Most users would not be able to easily translate that date so the BBjInputD control provides the mapping to display `2454628` as `June 10, 2008`. Likewise, you could display two-character state codes as their full name where NM becomes New Mexico.

Mapping in Action

Figure 1 and **Figure 2** show the same basic application, however, **Figure 2** uses the new mapping functionality to change the data in the CUSTOMER table of our Chile Company database into a more user-friendly form. We used the STATE table to display the corresponding full name for the two-character state codes stored in the CUSTOMER table. We perform a similar trick to convert the preferred shipping method from the five-character code stored in the CUSTOMER table to the full description.

continued...

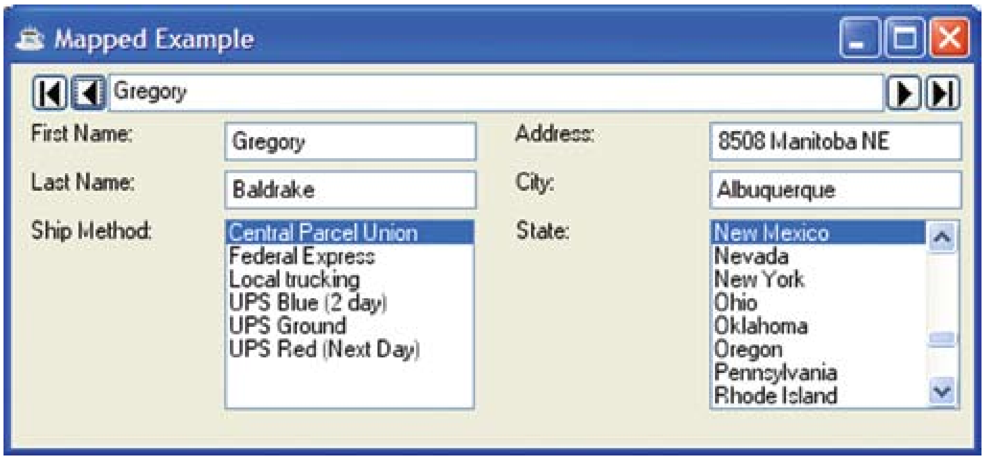


Unmapped Example

Gregory

First Name:	Gregory	Address:	8508 Manitoba NE
Last Name:	Baldrake	City:	Albuquerque
Ship Method:	CPU, FEDEX, TRUCK, UPBLU, UPGR, UPRED	State:	NM, NV, NY, OH, OK, OR, PA, RI

Figure 1. Application shows the raw data in the database

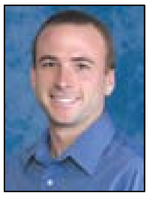


Mapped Example

Gregory

First Name:	Gregory	Address:	8508 Manitoba NE
Last Name:	Baldrake	City:	Albuquerque
Ship Method:	Central Parcel Union, Federal Express, Local trucking, UPS Blue (2 day), UPS Ground, UPS Red (Next Day)	State:	New Mexico, Nevada, New York, Ohio, Oklahoma, Oregon, Pennsylvania, Rhode Island

Figure 2. Application shows the meaningful data by using recordset mappings



Jon Bradley
Software Engineer

Furthermore, when the user changes the state or shipping method in the GUI front end, the data converts automatically from the full name back to the compact two-character state code. Before the addition of the mapping functionality, the application in **Figure 2** would have required either storing the full descriptions in the CUSTOMER table or writing the many lines of code required to handle the retrieval, mapping, display, and modification of the data.

Shut the Door on Database Changes

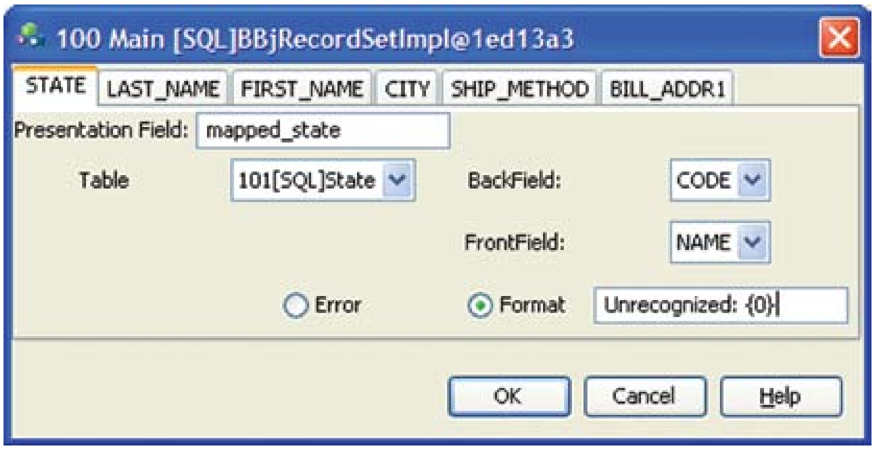
Changing the database, the alternative to using recordset mappings, is a bad solution. Since databases use foreign keys to reduce size, increase data normalization, and increase referential integrity, throwing them out just to display the description corresponding to that key is not a worthwhile consideration. Mappings provide an easy way to display the human readable descriptions without adding cumbersome code or denormalizing your database.

Open the Door to Mapping

Adding mappings to your application is simple; it introduces a new field to the recordset that becomes the field name for data binding a control.

A mapping consists of the value source and the mapping action. The value source is typically a BBjRecordSet that specifies how the database values should be mapped to presentation values and vice versa. The mapping action specifies what to do when the value source does not provide a valid mapping for a value.

The Chile Company database already contains a table STATE, which maps two-character state codes to full names. The database also has a table SHIP_METHOD that maps the five-character shipping codes to their full names. FormBuilder provides a dialog (**Figure 3**) to add mappings to a BBjRecordSet, the data from which the FormGen Wizard can then create the application. BASIS also expanded the API with new methods to add mappings.


100 Main [SQL]BBjRecordSetImpl@1ed13a3

STATE | LAST_NAME | FIRST_NAME | CITY | SHIP_METHOD | BILL_ADDR1

Presentation Field: mapped_state

Table: 101[SQL]State

BackField: CODE

FrontField: NAME

Error [✓ Format] Unrecognized: {0}

Figure 3. Mapping Editor Dialog in FormBuilder

If the number of possible choices for a field is relatively small, you may choose to build an in-memory BBjRecordSet to serve as your mappings.

Given a file like this:

Employee Table

NAME	ADJECTIVE CODE	HAS CAR CODE	JOB CODE
Jon	A	Y	1
Dave	B	Y	2
Tom	C	N	3
Frank	A	N	4

Where the Job Codes stand for the following:

Job Code Table

JOB CODE	JOB DESCRIPTION
1	Secret Agent
2	Key Grip
3	Unemployed
4	(NO MAPPING)

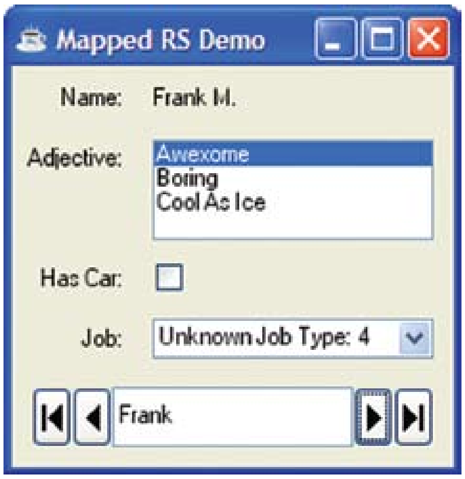
continued...

The code sample excerpt in **Figure 4** (download the complete sample file `MappedRecordset.src` noted at the end of this article) creates a mapping and binds a control to the new mapped adjective field.

```
TEMPLATE$="NAME:C(16*=0),ADJECTIVE:C(1),HAS_CAR:C(1),JOB_CODE:U(1)"
RecordSet! = BBJAPI().createFileRecordSet(FILENAME$,MODES$,TEMPLATE$)

REM MAP FOR JOB
jobMapRS! = BBjAPI().createMemoryRecordSet("CODE:U(1),DESC:C(16*=0)")
Data! = jobMapRS!.getEmptyRecordData()
Data!.setFieldValue("CODE", "1")
Data!.setFieldValue("DESC", "Secret Agent")
jobMapRS!.insert(Data!)

Data! = jobMapRS!.getEmptyRecordData()
Data!.setFieldValue("CODE", "2")
Data!.setFieldValue("DESC", "Key Grip")
jobMapRS!.insert(Data!)

Data! = jobMapRS!.getEmptyRecordData()
Data!.setFieldValue("CODE", "3")
Data!.setFieldValue("DESC", "Unemployed")
jobMapRS!.insert(Data!)

REM ADD THE MAPPING
sr! = RecordSet!.createRecordSetMappingSource(jobMapRS!,"code","desc")
ac! = RecordSet!.createFormatMappingAction("Unknown Job Type: {0}")
RecordSet!.addMapping("job_code","mapped_job",sr!,ac!)

REM Bind a control to mapped_job
myWindow!.addStaticText(104,10,200,70,25,"Job")
job! = myWindow!.addListEdit(204,100,200,125,100,"")
job!.fillFromRecordSet(jobMapRS!,"desc")
job!.bindRecordSet(RecordSet!,"mapped_job")
```

Figure 4. Excerpt from `MappedRecordset.src`

In **Figure 5**, the job field displays a formatted message, as specified by the mapping action, because the value source does not specify a mapping for the value 4.



Name: Frank M.

Adjective: Awexome
Boring
Cool As Ice

Has Car:

Job: Unknown Job Type: 4

Frank

Figure 5. The programmatic example application

Summary

The new mapping functionality makes it easier to use BBjRecordSets in normalized databases. The mappings make displaying human readable descriptions of table data easy, whether it is done in AppBuilder or via the BBj API. BASIS continues to make improvements in the mapping system and encourages developers to use the latest release to get the most functionality and speed from this new feature. So give mapping a warm welcome into your applications.

Download the code sample at
www.basis.com/advantage/mag-v12n1/mappedrecordset.zip

For more information about recordset mappings, read
Working With RecordSet Mappings
www.basis.com/solutions/RecordSetMappings.pdf