

Working with RecordSet Mappings

April 2008

By

Jon Bradley

The purpose of this guide is to introduce developers to RecordSet Mappings.

[Introduction](#)

[Terminology and Example Setup](#)

[BBJMappingSource](#)

[BBJMappingAction](#)

[Chronology: When the mappings happen](#)

[Adding Mappings with FormBuilder](#)

Working with RecordSet Mappings

Introduction

[Back to top](#)

A RecordSet is basically a pointer into a file or SQL result set. A RecordSet for a given template or result set has a collection of fields, which in turn, have a value at the current location of the RecordSet. The RecordSet can be moved through the backing data source. A GUI control can be bound to a field of a RecordSet. This allows the GUI to display the values within the recordset with no required support from the programmer. For more information on RecordSets, read these articles published in past issues of the *BASIS International Advantage*:

- [Using the BBjRecordset](#)
- [Why Use the BBjRecordSet?](#)

Sometimes the information on disk is not in the format that the developer wants to display to the user. For example, the database contains two-character state codes (NM, AZ, WY,...) but the developer would like to display its full state name (New Mexico, Arizona, Wyoming,...). To address this, BBj 8.0 adds the facility to map a field's value.

Terminology and Example Setup

[Back to top](#)

Suppose there are two SQL Tables – **Users**, and **LocationMap** – that look like this:

Users

NAME	LOCATION
John Doe	1
Jane Smith	3
Jane Doe	2
Dr. Droopy	0

LocationMap

CODE	DESC
0	NORTH
1	SOUTH
2	EAST
3	WEST

In the program, we have created two RecordSets:

```
UserRS! = BBjAPI().createSQLRecordSet(connect$,modes$, "SELECT NAME, LOCATION from USERS"); rem ( line 1)
LocationMap! = BBjAPI().createSQLRecordSet(connect$,modes$, "SELECT CODE, DESC from LOCATIONMAP"); rem( line 2)
```

Now let's create a mapping that utilizes the values in LocationMap as a mapping for the LOCATION field of Users:

```
source! = UserRS!.createRecordSetMappingSource(LocationMap!, "CODE", "DESC"; rem (line 3)
action! = UserRS!.createErrorMappingAction(); rem(line 4)
UserRS!.addMapping("LOCATION", "MAPPED_LOC", source!, action!); rem(line 5)
```

Let's examine the signature of `addMapping`.

```
void addMappingAction(string diskField,  
    string presentationField,  
    BBJMappingSource mappingSource,  
    BBJMappingAction mappingAction)
```

Note: BASIS coined and defined the terms *disk field*, *presentation field*, *mapping source*, *mapping action* below for use throughout the online documentation.

Parameters

Disk field: Specifies the field ("LOCATION") in the RecordSet (UserRS!) which we are trying to map. The term *disk field* describes the field in the RecordSet generically, regardless of the actual implementation of the backing store (SQL, File, Memory, etc).

Presentation field: Specifies the new field ("MAPPED_LOC") to add to the RecordSet(UserRS!) that populates with the mapped value. The term *presentation field* describes the field that is a result of a mapping.

Mapping source: Specifies the data source for the mapping. Attempting to populate the *presentation field* consults the mapping source first. If the mapping source cannot produce a mapping for a given value, it executes the mapping action.

Mapping action: Specifies the *mapping action* to be taken if the value source does not provide a mapping, the mapping action is executed.

BBJMappingSource

[Back to top](#)

Let's look at the signature of `createRecordSetMapping`, used in line 1:

```
BBJMappingSource createRecordSetMapping(BBJRecordSet mapRS,  
    string backField,  
    string frontField)
```

Parameters

MapRS: Specifies the RecordSet to consult for values when attempting to map a value of a disk field into a value for a presentation field.

Back field: Specifies the field in a mapRS to be used as a key when attempting to map a value from the disk field to the presentation field. The RecordSet mapping system uses the backfield as the value when mapping from the presentation field back to the disk field if the presentation field is modified and flushed back to disk via an update.

Front field: Specifies the field in mapRS to be used as the value when attempting to map a value from disk field to the presentation field. Conversely, the RecordSet mapping system uses the front field as the key when mapping from the presentation field back to the disk field.

The BASIS term *forward mapping* is the process of mapping a value in the disk field to a value for the presentation field. Similarly, *backward mapping* is the process of mapping a value in the presentation field to the disk field.

When forward mapping, the BBJRecordSetMappingSource only provides a mapped value for the disk field's value "X" if there is one and only one row with "X" in the back field.

When backward mapping, the `BBjRecordSetMappingSource` only provides a mapped value for the presentation field's value "Y" if there is one and only one row with "Y" in the front field.

A `BBjEmptyMappingSource` provides no mapping and thus allows execution of the mapping action.

When using `SQLRecordSets` as the mapping source for a `BBjRecordSetMappingSource`, your query must take the simple form "SELECT <LIST> from TABLE".

BBjMappingAction

[Back to top](#)

The `BBjMapping` action specifies what to do if the mapping source fails to provide a mapping. In our current example, we created a `BBjErrorMappingAction` because we know the entire set of values for "LOCATION." Anything else is a serious error.

In another situation, the programmer may choose to display a message as some function of the value in the disk field. The `BBjFormatMappingAction` takes in its constructor a format string such as "Unknown Value: {0}" and produces a mapping of "Unknown Value: <diskVal>." The format string need not contain a {0}, thus, it can produce a generic display message. Refer to the online documentation [BBjRecordSet::createFormMappingAction](#) for more information.

Chronology: When the mappings happen

[Back to top](#)

The value mappings and possible subsequent errors always occur in response to a programmatic action. A forward mapping occurs whenever the `RecordSet` moves or the mappings change. A backward mapping occurs whenever the `RecordSet` is saved to the backing store via `BBjRecordSet.update()` if the presentation field has been modified more recently than the disk field.

Adding Mappings with FormBuilder

[Back to top](#)

A developer can add mappings to a `RecordSet` in `FormBuilder` via the Edit Mappings Action with a right click on a `RecordSet` node in the `BBjGui Inspector`.

The API documentation contains a number of sample programs related to `RecordSet` mapping.

- [BBjRecordSet::addMapping](#)
- [BBjRecordSet::getMappingDescription](#)
- [BBjRecordSet::createRecordSetMappingSource](#)