

Record Objects Ride to the Rescue



Heikki Malmberg

The Problems

- Running out of fields in critical file - DETM
 - About 3500 fields used
 - File holds entitlement information for customers
 - Separate field group for each benefit component
 - ▶ eg Pensions Rental Assistance, Pensions Pharmaceutical Allowance, Parenting Basic
 - Design based on performance considerations many years ago
 - New project would have exceeded 4000 field limit
- Average record length 7,500 with std dev 10,000
 - significant field scan cost

The Solution

- Split file into group with multiple files
 - each file has own set of fields
 - require access as a file group for common software
 - and to make things more interesting:
 - ▶ slightly different record structure (extra field, and change in field format)
 - ▶ standardise field names
 - not all fields moving off DETM file
- Need to change existing software owned by many different application areas
 - But in a way that doesn't break everything while this is happening
 - Don't like Big Bang changes

Two Options

- FRN in MEMBER
 - Requires carrying around two variables
 - Used this in the past and not entirely satisfactory
 - ▶ more data access api's
- Sirius record objects
 - enables file details to be hidden inside class
 - can use properties to hide differences in record fields
 - enables virtual longstring fields (via properties)
 - cancels request if object null rather than retrieve record 0
 - can control record locking behaviour centrally

Comparison - Old code

```
%detm.irn is string len 8
...
call dr.detm.get.irn (%crn, %detm.irn)
...
if %detm.irn ne '' then
    in '?&FPD'DETM -
    frn %detm.irn
        *... Read data
    end for
end if
```

Comparison - New OO code

```
%detmRec is object dam:rec:detm
...
%detmRec = get(%crn, 'PEN')
...
if %detmRec is not null then
    for record %detmRec:recObj
        *... read data
    end for
end if
```

Our Record class

- Embeds Sirius record object in our record object (more about this later)
- Hides the actual physical file from application
 - No need to dummy string '?&FPD' substitution in application code
 - ▶ use dummy strings to support multiple test/dev environments
- In general this technique will enable us to change from file to group with no or minimal changes to application, eg if we run out of IRN's
- Record structure simple
 - one record per customer per benefit

Class declaration

```
class dam:rec:detm
  public
    variable recObj          is object record in '?&FPD'GDT
    variable benefit         is string len 3
    property hintList        is longString

    constructor get(%crn      is string len 9, -
                   %benefit  is string len 3)

    constructor prep(%crn     is string len 9), -
                   %benefit  is string len 3)

    ...
  end public
end class
```

Class declaration - notes

- *benefit* variable held redundantly as accessed often
 - but could convert into a property if we change our mind
- *hintList* treated as longstring but held in database as a repeating field
- *get* constructor retrieves existing database record
- *prep* constructor retrieves existing record if exists, otherwise creates one

Other changes

- Converted old field group data update subroutines to OO classes with update and read methods
 - generated based on the same Repository metadata
 - retrieval functions return stringlists (with images) so no direct access to native fields required (internally using \$field_listi)
 - use optional and named parameters
 - ▶ much easier to implement subsequent changes, eg June release
- Rationalised the old modules
 - reduced from over 100 to 50
 - cleaned up differences that have crept in over time

Three groups for developers

- Numerous application areas (eg Pensions, Families)
 - own the code for entitlement calculation
- CAPS (Customer Assessment Payment Support)
 - own common software used by applications
 - use the stringlist OO functions so no direct field access
 - field names defined in incore tables
- Database Design team
 - we provide data access classes - records and field groups

The cunning plan

1. Write OO classes for record and data access
 - using the old DETM file structure
2. Convert CAPS common code to use OO
3. Run regression tests (including TPNS) to ensure CAPS and database software OK
 - If everyone changed software at the same time, its a lot harder to isolate problems
4. Convert rest of applications to use OO classes and test
 - These could be independently tested
5. Change data access methods to operate with mixture of old DETM and new files

The cunning plan (cont)

6. Switch application one by one to new split files (using incore table entries)
 - deal with field name changes as they occur
7. Change data access methods to operate only from split files
8. Test everything
9. Proceed with other release changes

Issue 1 - Undefined fields in group context

- Access to a field that exists in a file in a group but not the one being accessed does NOT cause request cancellation
 - null is return and on update the data disappears
 - not good especially when dealing with entitlement data
- Solution - Sirius changed behaviour of record objects to cancel request for these undefined field references (Sirius mods 7.0)

Issue 2 - Record locking

- **FRN uses shared lock for duration of FRN loop**
- **Record objects obtain lock on instantiation and released when object discarded**
 - so even though the syntax looks similar the record locking behaviour is quite different
- **Decided that no record locking was safer than shared locks**
 - eg batch job creates record object for CRN and does not discard it
- **Sirius mods 7.0 has new record locking technique - loopLockStrength**
 - Works similar to FRN - shared lock only obtained during 'for record' processing
 - Very slight performance hit but comparable to FRN
 - Will implement this in June (have run through system and TPNS testing)

Issue 3 - Embedded record object

- In Sirius mods 6.9 not possible to inherit from record object
- Hence our record object contains the Sirius record object:
 - for record %detmRec:recObj
- This is fine and works, but inheritance is cleaner
- Sirius mods 7.0 enables inheritance, hence new code:
 - for record %detmRec

Issue 3 - New class declaration

```

class dam:rec:detm extends record in '?&FPD'GDT inherit
  public
*   variable recObj           is object record in '?&FPD'GDT
    variable benefit         is string len 3
    property hintList       is longString

    constructor get(%crn      is string len 9, -
                    %benefit is string len 3)

    constructor prep(%crn     is string len 9), -
                    %benefit is string len 3)

    ...
  end public
end class

```

Production release

- Released in March 2007
 - No issues have arisen
- CPU saving of 2.5% (minimum) from splitting the file and reducing field scans
 - possibly more like 4% as other changes, such as increased use of \$field_listi, couldn't be factored out (overall the release saved 10%)
- Equivalent to about 500 (800?) MIPS
- After split average record length for remaining DETM file 4,700 with std dev 6,500

Sirtune comparison

Pre March

1	SBNU	22.979	22.979	<< - field extract module
2	DKBM	16.216	39.195	
3	EVNU	7.814	47.010	

Current

1	DKBM	17.253	17.253	
2	SBNU	17.231	34.485	<<- field extract module
3	EVNU	7.526	42.011	