

Data Structures and Data Structure Arrays

Making them work for you

Brian May
IBM i Modernization Specialist
Profound Logic Software

Webmaster and Coordinator
Young i Professionals

Overview

- Discuss basics of RPG data structures
- Discuss QUALIFIED data structures
- EVAL-CORR
- Sorting data with data structures and arrays
- Discuss data structure arrays
- Data structure templates
- Complex data structures

Data Structure Basics

What every RPG programmer should know

The Basics – Format of a DS

```
dDS_Name          ds
d Subfield1        5
d Subfield2        2 0
d Subfield3        Like(Field3)
```

- The Name of the data structure should be listed on the first line of the definition
- Remaining lines define the subfields within the data structure
- Subfields can be defined explicitly or based on other fields

The Basics – DS uses

```
dDS_Name      ds
d Smart_Item_Num    1    15
d Mfg_Location      1    3
d Division          4    6
d Item              7    15
```

- Subfields can be overlayed to combine and/or dissect fields
- Optional format:

```
dDS_Name      ds
d Smart_Item_Num    15
d Mfg_Location      3    Overlay(Smart_Item_Num)
d Division          3    Overlay(Smart_Item_Num:4)
d Item              9    Overlay(Smart_Item_Num:7)
```

The Basics – DS uses

```
dCustomer_Data  ds
d First_Name    15
d Last_Name     15
d Email         50
d Address1      50
d Address2      50
d City          15
d State         2
d Zip           10
d Phone         15
```

- Group related data
- Data grouped together in a data structure can be easily passed as a single parameter

The Basics – Special DSs

- PSDS – Program-Status Data Structure
 - Identified by an “S” in position 23 of first data structure line.
 - Special data structure containing program exception/error information
- INFDS – File Information Data Structure
 - Identified by using the INFDS() keyword on a file specification
 - Special data structure containing file exception/error and feedback information. Can be defined for each file used.

The Basics – Special DSs

- Data Area Data Structure
 - Identified by a “U” in position 23 of first data structure line.
 - Special data structure that automatically reads in and locks the data area of the same name at program start and outputs and releases the data area on program termination.
- INDDS – Indicator Data Structure
 - Identified by the INDDS() keyword on a file specification
 - Special data structure containing the 100 possible indicators used by the display or print file

Qualified Data Structures

The extra typing is not wasted

What is a QUALIFIED DS?

- Qualified data structures instruct the compiler to require the data structure name whenever a subfield is referenced
 - Example: `Customer.Email`
- Qualified data structures can have subfields with the same name as other fields without name collision
- Qualified data structures are required when using certain data structure features, such as data structure arrays

How to Create a Qualified DS

- QUALIFIED keyword
 - Simplest way to make a qualified data structure

```
dCustomer      ds      Qualified
d First_Name    15
d Last_Name     15
d Email         50
d Phone         15
```

How to Create a Qualified DS

- LIKEDS keyword
 - LIKEDS creates a data structure with the same subfields as another data structure
 - A data structure defined with LIKEDS is automatically created as qualified

```
dCustomer      ds
d First_Name    15
d Last_Name     15
d Email         50
d Phone         15

dNew_Customer   ds      LikeDS (Customer)
```

How to Create a Qualified DS

- LIKEREK keyword
 - LIKEREK creates a datastructure with subfields matching the column names and formats for a specified record format
 - A data structure defined with LIKEDS is automatically created as qualified

```
fItemMast  if  e          k Disk
dItem_Data      ds          LikeRec (ItmMastRec)
```

Using Qualified DSs

- All references to subfields must have the data structure name, a period, and then the subfield name.

```
New_Customer.First_Name = 'Brian' ;
New_Customer.Last_Name = 'May' ;
New_Customer.Email = 'brian@youngiprofessionals.com' ;
New_Customer.Phone = '6625551212' ;
```

Using Qualified DSs

- EVAL-CORR operation code
 - Sets subfields with the same name and compatible data types equal
 - Saves monotonous code to set each subfield equal to its corresponding field in the other data structure

```

dCustomer      ds
d First_Name   15
d Last_Name    15
d Email        50
d Phone        15

dProspect      ds
d First_Name   25
d Last_Name    25
d Phone        10
d Address1     25
d Address2     25
d City         15
d State        2
d Email        30

/Free
Eval-Corr Customer = Prospect ;

```

Other Cool Stuff

Never could figure out what to call this section....or even where it belonged

Another Way to Make DS Like a File

- EXTNAME() keyword
 - Creates an UNQUALIFIED data structure with the same subfields as the file referenced
 - Externally defined data structures require an “E” in position 22 of the definition specification

```
fItemMast  if  e          k Disk
dItem_Data      e ds          ExtName('ITEMMAST')
```

Changing Externally Defined Fields

- EXTFLD keyword
 - Used to identify a externally defined field
 - You can change attributes of an externally defined field
 - Change the name
 - Change the default INZ value
 - Divide the field into an array

```
dItem_Data      e ds          ExtName('ITEMMAST')
d Item_Number   e          ExtFld(ITMNO)
d Status        e          Inz('I')
d SizFlgs       e          Dim(10)
```

Changing Externally Defined Fields

- PREFIX keyword
 - Works the same as the PREFIX keyword for file specifications
 - Allows you to keep names the same with the file
 - Not allowed with LIKERECD(). LIKERECD() includes prefixes specified on the file specification

```
fItemMast  if  e          k Disk  Prefix(IM_:2)
dItem_Data      e ds          ExtName('ITEMMAST')
d              Prefix(IM_:2)
```

A Cool Data Sorting Trick

- This procedure will allow you to store arrays of related data in a data structure and sort by any given subfield
- First off, you need to define an array large enough to hold all of the data in your “record”
- Then overlay individual fields on top of the array

```
dSalesDS      ds
d SalesArray          75  Dim(100)
d Region            10  Overlay(SalesArray)
d Salesman          25  Overlay(SalesArray:*Next)
d Item              25  Overlay(SalesArray:*Next)
d Quantity          15s 0 Overlay(SalesArray:*Next)
```

A Cool Data Sorting Trick

- Load your data into the data structure however you like
- Keep up with how many “records” you add
- Use a combination of SORTA and %SUBARR to sort the data you have loaded
- This will cause all of the data to be sorted by the subfield selected

```
SortA %SubArr (Region:1:NumRecs) ;
```

Data Structure Arrays

This has been around for a while.....So stop using MODS already

An Array of Data Structures?

- Why on earth would I want that?
 - Useful for holding multiple database records in memory
 - Can dramatically improve performance by avoiding file i/o looking for commonly found records
 - Eliminates the need for temporary table in QTEMP in some cases
 - Makes multidimensional arrays possible
 - It is easy to set up and use

Creating a DS Array

- The data structure must be QUALIFIED
- The DIM() keyword is added to the DS definition just like it would be used on a field
- To reference a subfield in a DS array, you must use the format: DS_Name(Index).Subfield

```

dCustomers      ds      Qualified Dim(50)
d First_Name    15
d Last_Name     15
d Email         50
d Phone         15

/Free

Customers(1).First_Name = 'Brian' ;
Customers(1).Last_Name = 'May' ;
Customers(1).Email = 'brian@youngipprofessionals.com' ;
Customers(1).Phone = '6625551212' ;

```

Array of Database Records

- Create your DS using LIKERECS
- Add the DIM keyword
- Now I can hold lots of records in memory

```
fStoreMast if e k Disk
dStore_Recs ds LikeRec (STORERECS)
d Dim(50)
```

Multidimensional Arrays

- Sometimes you need to arrange data in more than one dimension
- By putting an array as a subfield of a data structure array, you now have the ability for 2 dimensions
- In our example of students in each class period, I could make myself the 5th student in the 3rd period class like this:
 - Classes(3).Student_Name(5) = 'Brian May' ;

```
dClasses ds Qualified Dim(8)
d Student_Name 50 Dim(30)
```

Complex Data Structures

You mean it gets more complex than this?

What is a Complex DS?

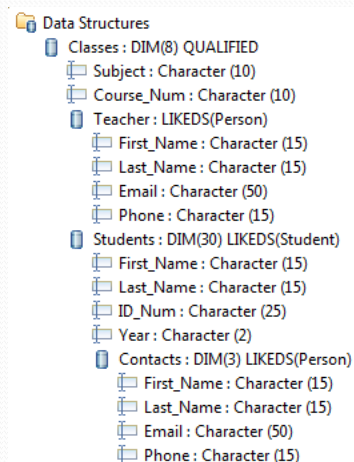
- Complex data structures are basically, data structures inside of data structures
- Complex data structures are always QUALIFIED
- In order to define a data structure inside another data structure, we use LIKEDS
- Data structure arrays can be used in complex data structures next to subfields and other arrays

Why Would I Want to Do That?

- Allows for all of your related data to be organized and grouped in a logical manner
- Very useful when using XML-INTO to parse an XML document in RPG
- Great for passing a large amount of interrelated data between procedures or programs

Building a Complex DS

- Lets expand our Classes data structure and add more detail
- Here is an overview of the structure as seen in the outline view in RDi



Building a Complex DS

- The TEACHER of the class and the CONTACTS for the student are both based on a DS called PERSON
- The STUDENTS in the class are based on a DS called STUDENT

```

dPerson      ds      Based (Empty)
d First_Name 15
d Last_Name  15
d Email      50
d Phone      15

dStudent     ds      Qualified Based (Empty)
d First_Name 15
d Last_Name  15
d ID_Num     25
d Year       2
d Contacts   LikeDS (Person) Dim (3)

```

Building a Complex DS

- Neither of these data structures will ever actually be used. They are templates
- To prevent storage from being allocated, I used the BASED keyword on them with an undefined pointer
- If you are on i 6.1 or higher, you can use the TEMPLATE keyword instead
- The STUDENT data structure had to be defined as QUALIFIED because it has a LIKEDS subfield
- The CONTACTS data structure array is defined as DIM(3) to allow up to 3 emergency contacts

Building a Complex DS

- Now we just need to define our CLASSES data structure and base some of our subfields on our templates
- We are allowing 8 classes with one teacher and 30 students
- Pretty cool, huh?

```

dClasses      ds      Qualified Dim(8)
d Subject          10
d Course_Num      10
d Teacher          LikeDS(Person)
d Students         LikeDS(Student) Dim(30)
  
```

Using a Complex DS

- Each level of the data structure needs to be referenced and separated by a period until you get down to the level and subfield you want
- Obviously, names can get very long. This is where auto-complete in RDi can really save you some keying

```

Classes(1).Course_Num = 'ENG1045' ;
Classes(1).Teacher.Email = 'mrsmith@school.edu' ;
Classes(1).Students(7).Contacts(1).First_Name = 'John' ;
Classes(1).
  
```

32E The Subject Course_Num Teacher Students Subject is not char(10)

07E A of a free-form

Questions?

Thank You!

About the Presenter

Brian May is an IBM i Modernization Specialist for Profound Logic Software. He is also webmaster and coordinator for the Young i Professionals (<http://www.youngiprofessionals.com>). He is a husband and father of two beautiful girls. Brian can be reached at bmay@profoundlogic.com

