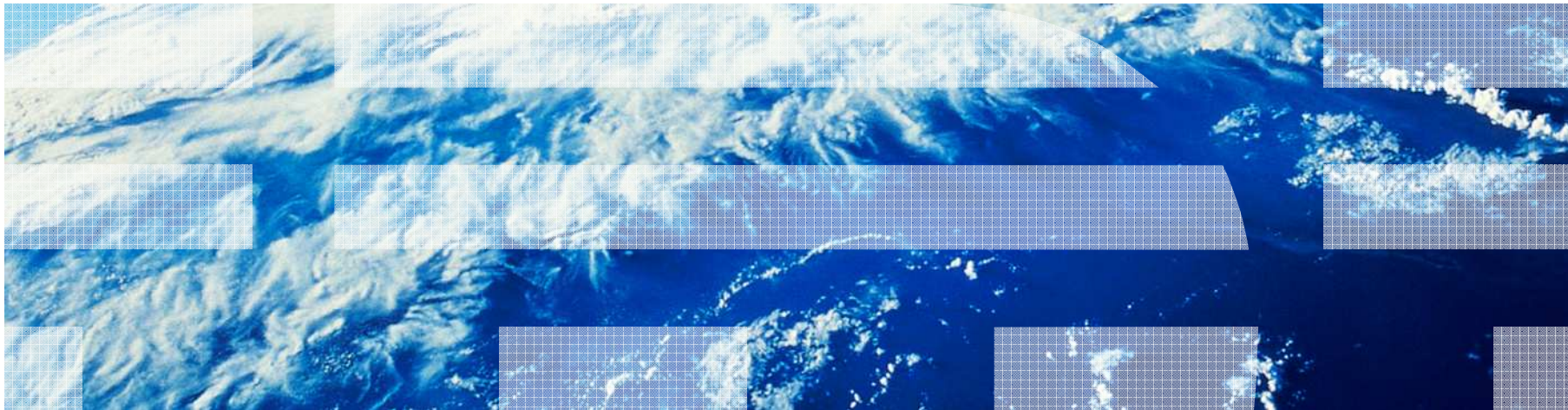# *IRRXUTIL: Analyzing your RACF Database Using REXX*

Mark Nelson
SHARE Session 8255
z/OS Security Server (RACF) Design and Development. IBM Poughkeepsie
markan@us.ibm.com

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Agenda

- **What is R_admin?**
  - Functions
  - Authorizations

- **What is IRRXUTIL?**
  - Relationship to R_admin
  - Invocation syntax
  - Sample invocation
  - Considerations
  - Return Codes
  - Returned data
  - Getting the "next" profile
  - Finding field names
  - References

What is the R_admin Callable Service?

- **The R_admin callable service (IRRSEQ00) is an assembler programming interface which allows for management of RACF profiles and system wide settings (SETROPTS)**

- **R_admin allows you to:**

  - Execute RACF commands
    - With the exception of RVARY, BLKUPD, RACLINK, RACF operator commands (TARGET, SET, SIGNOFF, etc.)

  - Update/Extract profile information into a tokenized format
    - USER, GROUP, user-to-group connections, general resources including access lists
    - Data set profiles (UPDATE only)

  - Set/Extract SETROPTS information
    - SMF Unload-like format
    - "Tokenized" format

  … and more!

Authorization for R_admin

- **R_admin may be invoked by authorized and unauthorized callers.**

  – Authorization is required to set or change the user ID under which the function is performed.

  – Non-authorized callers cannot use the R_admin update function codes

  – Non-authorized callers must have READ authority to a function-spepcific resource in the FACILITY class. For example:

    • IRR.RADMIN.command for a RACF command (such as IRR.RADMIN.LISTUSER for an LU command)

    • IRR.RADMIN.SETROPTS.LIST to extract SETROPTS data

What is IRRXUTIL?

- **IRRXUTIL allows a REXX program to use the R_admin interface to extract RACF profile and SETROPTS Data**

  - Supports the extraction of USER, GROUP, CONNECT, RESOURCE and SETROPTS data from RACF
  - Data set extraction not supported
  - Digital Certificate information not supported

- **IRRXUTIL places the returned data directly into REXX variables which can then be easily used simply by referencing the REXX variables**

- **Since IRRXUTIL uses R_admin, you must authorize IRRXUTIL users to the underlying R_admin function**

What IRRXUTIL is not

- IRRXUTIL does not have any support for any of the other function codes supported by R_admin, such as those which update profile information

- However, it is relatively simple to create a command invocation and run it directly from REXX. Certainly simpler than attempting to create any sort of REXX data structure to map back the tokenized functions of R_admin.

- Because R_admin does not support the extraction of data from RACF DATASET profiles, IRRXUTIL does not support RACF DATASET profiles.

IRRXUTIL Invocation Syntax

- **myrc=**IRRXUTIL(*function,type,profile,stem,prefix,generic*)

  - *Function:* "EXTRACT" or "EXTRACTN"

  - *Type:* "USER", "GROUP", "CONNECT", "_SETROPTS", *general resource class.* <u>DATASET not supported</u>.

  - *Profile:* Profile to extract. Case sensitive. Specify '_SETROPTS' for SETROPTS data.

  - *Stem:* REXX stem variable name to populate with results. Do not put the '.' at the end.

  - *Prefix:* Optional prefix for returned variable name parts (more later)

  - *Generic:* Optional, 'TRUE' or 'FALSE' (uppercase). Applies to general resource profiles only.

A Quick Example

- **Here is a simple program which retrieves a general resource profile and dumps the access list.**

```rexx
/* REXX */
myrc=IRRXUTIL("EXTRACT","FACILITY","BPX.DAEMON","RACF","","FALSE")
say "Owner: "RACF.BASE.OWNER.1
Say "ACL:"
do a=1 to RACF.BASE.ACLCNT.REPEATCOUNT
    Say "  "||RACF.BASE.ACLID.a||":"||RACF.BASE.ACLACS.a
end
```

```
READY
EX 'SAMPLE.CLIST(IRREXXRS)'
Owner: IBMUSER
ACL:
        IBMUSER:READ
        WEBSRVR:READ
        MEGA:READ
        LDAPSRVR:READ
        FTPD:READ
READY
```

- Note the complete lack of parsing code. Just retrieve the profile and directly access the required data.

- Note also the lack of return code checking. Bad code. No donut!

9

IRRXUTIL Considerations

- **The caller needs access to use R_admin extract via the appropriate FACILITY class profile protecting the desired function.**

- **In addition, the caller must be allowed to retrieve the profile in question. The caller will only have fields they are allowed to view returned.**

- **This is all enforced by the R_admin extract function which IRRXUTIL calls.**

Profile Type	Required FACILITY profile
User, Connect	IRR.RADMIN.LISTUSER
Group	IRR.RADMIN.LISTGRP
General Resource	IRR.RADMIN.RLIST
Setropts	IRR.RADMIN.SETROPTS.LIST

IRRXUTIL return codes

- myrc=**IRRXUTIL(*function,type,profile,stem,prefix,generic*)**

- MYRC is the return code from IRRXUTIL. It is a list of 5 numbers. If the first=0, IRRXUTIL was successful and data has been returned.

Description	RC1	RC2	RC3	RC4	RC5
Success	0	0	0	0	0
Warning, stem contained '.'	2	0	0	0	0
Bad number of parameters specified	4	Number of parms specified	Min number allowed	Max number allowed	0
Parameter Error	8	Index of bad parameter	1=Bad length 2=Bad value 3=Imcompatible with other parms	0	0
R_admin failure	12	12	R_admin safrc	R_admin racfrc	R_admin racfrsn
Environmental error	16	0=Rexx Error 4=R_admin error	For IBM support	For IBM support	0

Common Return Codes

- **0 0 0 0 0 = Success**

- **8 x y 0 0 = Error in IRRXUTIL invocation**
  - "x" – Number of the incorrect parameter
  - "y" – What's wrong
    - 1: Bad length
    - 2: Bad value
    - 3: Inconsistent with other parameters

- **12 12 4 4 4 = Profile not found**

- **12 12 8 8 24 = Not authorized to R_admin extract**

Return Code Checking

Check the first value in the return code string. If it is 0, the call was successful.

```rexx
/* REXX */
myrc=IRRXUTIL("EXTRACT","FACILITY","BPX.DAEMON","RACF","","FALSE")
If (word(myrc,1)>0) then do
    say "Error calling IRRXUTIL "||myrc
    exit
end
say "Profile name: "||RACF.profile
do a=1 to RACF.BASE.ACLCNT.REPEATCOUNT
    Say "   "||RACF.BASE.ACLID.a||":"||RACF.BASE.ACLACS.a
end
```

Two Ways to Process IRRXUTIL Output

- **The variables returned by IRRXUTIL can be used in 2 ways:**

  - Known data can be retrieved directly by simply referencing REXX variables by segment and field.

  - Programs with no knowledge of what segments and fields exist are given enough information to find all of the segments and fields returned by IRRXUTIL.

    - But, there is no mechanism to find out all *potential* segments/field which could exist. It only returns what exists for a given profile.

Retrieving Unknown Data Example

```
                                                                    stem.BASE.NAME
                                  stem.BASE
                                                                    .0 = 1
                                  .0 = 30                            .1 = "BRUCE WELLS"
stem.0 = 4                        .1 = "NAME"
     .1 = "BASE"                  .2 = "SPECIAL"                     stem.BASE.SPECIAL
     .2 = "TSO"                   .3 = "CLAUTH"                      .0 = 1
     .3 = "OMVS"                  .4 = … … …                         .1 = "FALSE"
     .4 = "CICS"
                                  .FLAGS = "00000000"                stem.BASE.CLAUTH

     .PROFILE = "BRWELLS"                                            .0 = 3
     .CLASS   = "USER"                                              .1 = "USER"
     .GENERIC = "FALSE"                                             .2 = "FACILITY"
     .VERSION = 0                                                   .3 = "UNIXPRIV"

                                                                    .BOOLEAN = "FALSE"
                                                                    .OUTPUTONLY = "FALSE"
                                                                    .REPEATING = "TRUE"
```
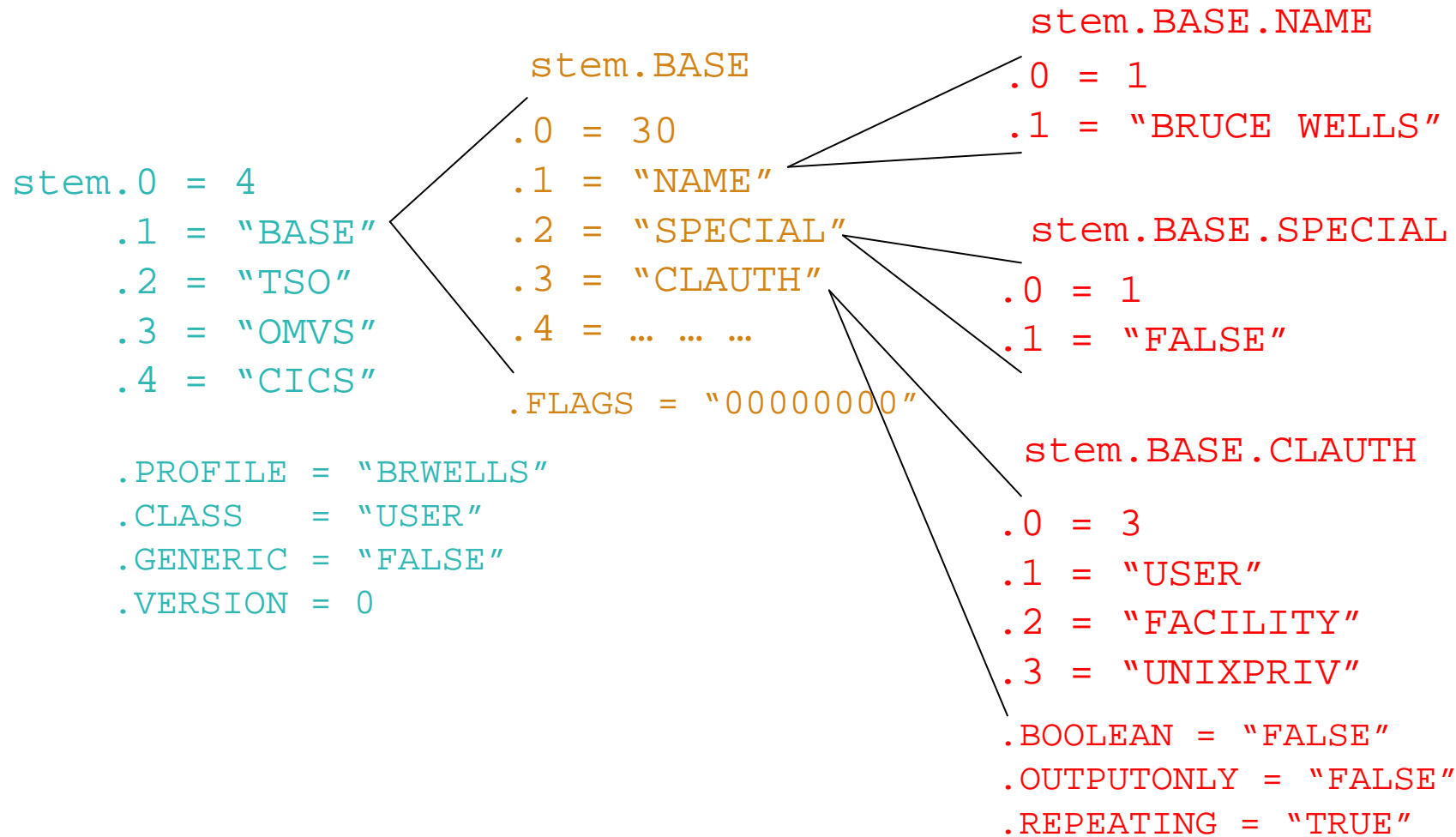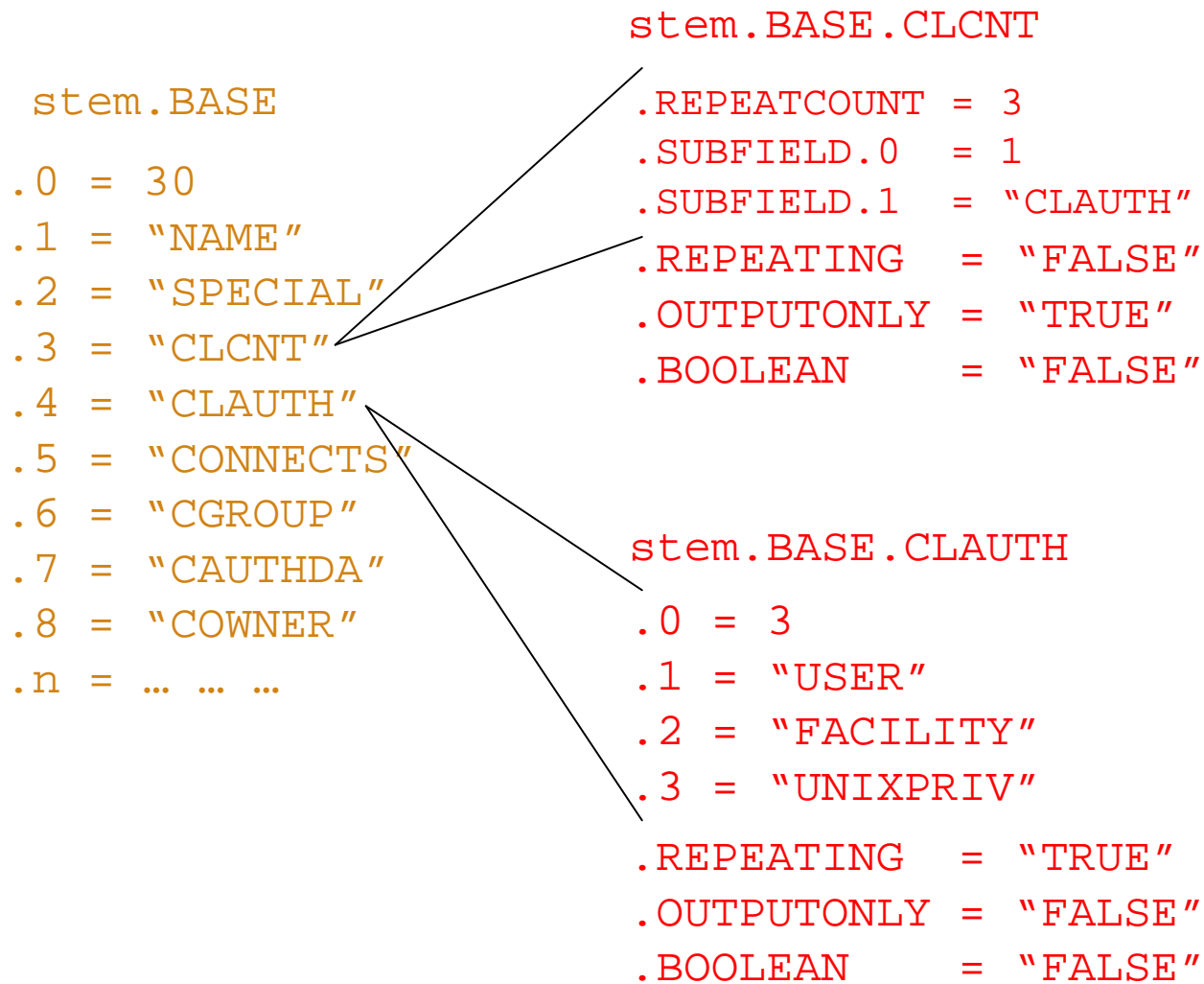
Retrieving Repeating Data

Repeating fields have some additional control information stored in the 'repeat header' field.

- **Stem.*segment.field*.repeatCount:** Non-zero value indictates *field* is a repeat header. This is the number of repeat groups for this field.

- **Stem.*segment.field*.subfield.0:** Number of subfields in this repeat group.

- **Stem.*segment.field*.subfield.1-n:** subfield names

- **Stem.*segment.subfieldname*.0:** same as Stem.*segment.field*.repeatCount. Number of values.
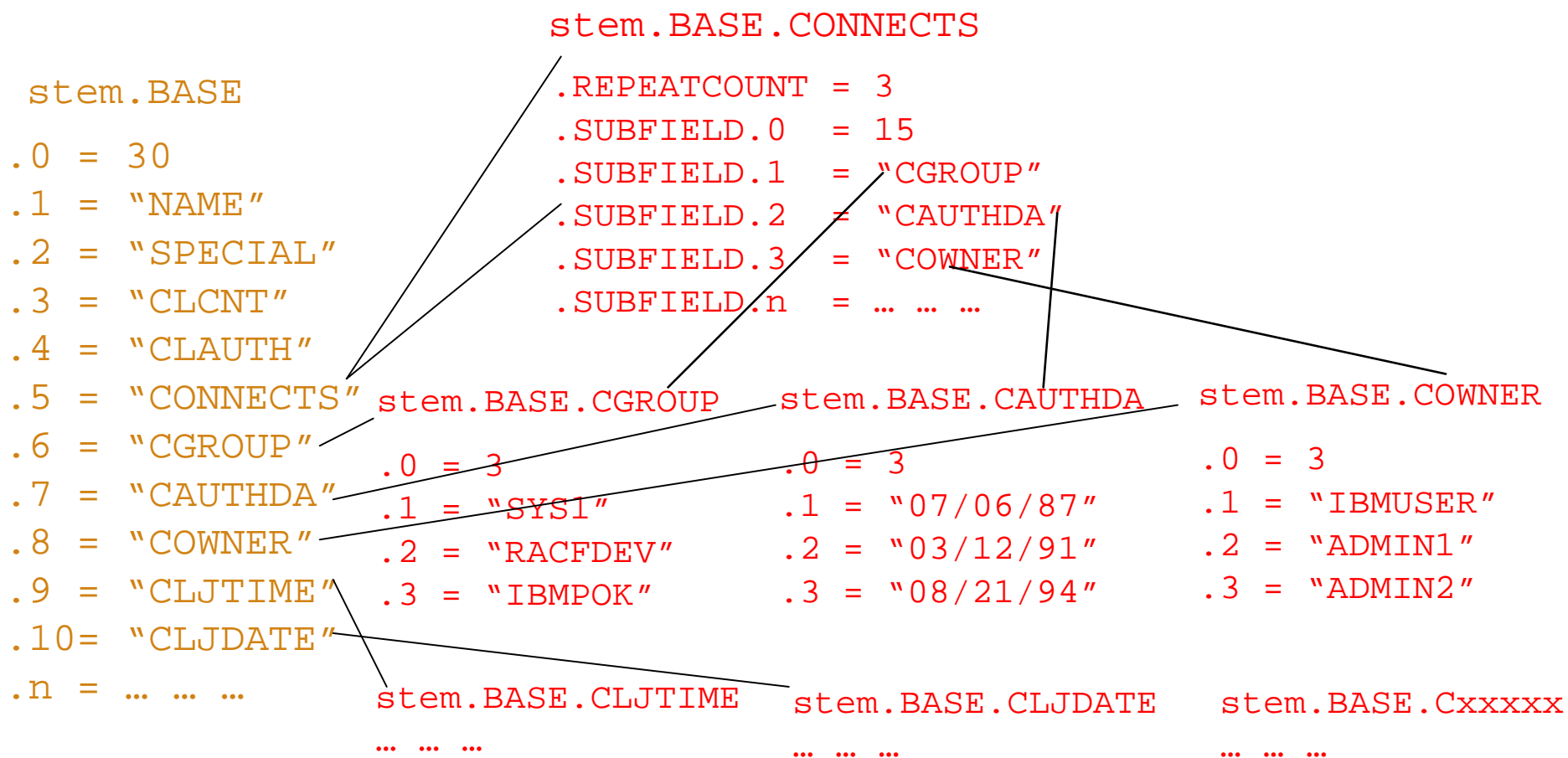
- **Stem.*segment.subfieldname*.1-n:** subfield values

Much needed example on next page

Stem Structure for a Simple Repeating Field

```
                                        stem.BASE.CLCNT

stem.BASE                               .REPEATCOUNT = 3
                                        .SUBFIELD.0  = 1
.0 = 30                                 .SUBFIELD.1  = "CLAUTH"
.1 = "NAME"                             .REPEATING   = "FALSE"
.2 = "SPECIAL"                          .OUTPUTONLY  = "TRUE"
.3 = "CLCNT"                            .BOOLEAN     = "FALSE"
.4 = "CLAUTH"
.5 = "CONNECTS"
.6 = "CGROUP"
.7 = "CAUTHDA"                          stem.BASE.CLAUTH
.8 = "COWNER"
.n = … … …                              .0 = 3
                                        .1 = "USER"
                                        .2 = "FACILITY"
                                        .3 = "UNIXPRIV"
                                        .REPEATING   = "TRUE"
                                        .OUTPUTONLY  = "FALSE"
                                        .BOOLEAN     = "FALSE"
```

Stem Structure for a Complex Repeating Field

```
                              stem.BASE.CONNECTS
 stem.BASE
                              .REPEATCOUNT = 3
.0 = 30                       .SUBFIELD.0  = 15
.1 = "NAME"                   .SUBFIELD.1  = "CGROUP"
.2 = "SPECIAL"                .SUBFIELD.2  = "CAUTHDA"
.3 = "CLCNT"                  .SUBFIELD.3  = "COWNER"
.4 = "CLAUTH"                 .SUBFIELD.n  = … … …
.5 = "CONNECTS"  stem.BASE.CGROUP    stem.BASE.CAUTHDA    stem.BASE.COWNER
.6 = "CGROUP"
.7 = "CAUTHDA"   .0 = 3              .0 = 3               .0 = 3
.8 = "COWNER"    .1 = "SYS1"         .1 = "07/06/87"      .1 = "IBMUSER"
.9 = "CLJTIME"   .2 = "RACFDEV"      .2 = "03/12/91"      .2 = "ADMIN1"
.10= "CLJDATE"   .3 = "IBMPOK"       .3 = "08/21/94"      .3 = "ADMIN2"
.n = … … …
                 stem.BASE.CLJTIME   stem.BASE.CLJDATE    stem.BASE.Cxxxxx

                 … … …               … … …                … … …
```

Sample Code to Get Field Names

- **Here's a code fragment which shows how to extract all of the field names for a profile**

```
/* REXX */
 RACF.profile='IBMUSER'
 myrc=IRRXUTIL("EXTRACT","USER",RACF.profile,"RACF")
     If (Word(myrc,1) <> 0) Then Do
         Say myrc
         exit
         end

 say "The USER profile" RACF.profile " has " racf.0
"segments"

/* Continued on the next page */
```

Sample Code to Get Field Names…

```
do i=1 to RACF.0            /* get the segment names */
    segment=RACF.i
    say "====" segment "===="
    do j=1 to RACF.segment.0
       field=RACF.segment.j
       say "   FIELD="LEFT(Field,8,' ')
       end
    end
```

- **The Result:**

```
The USER profile IBMUSER  has  2  segments
==== BASE ====
    FIELD=CREATDAT
    FIELD=OWNER
    FIELD=ADSP
    FIELD=SPECIAL
    FIELD=OPER
    FIELD=REVOKEFL
    FIELD=GRPACC
    …
```

The Importance of Prefixing

- **Consider the following program which determines if the OMVS UID of the supplied user id matches a supplied UID value.**

```
/* REXX */
arg user idNum
myrc=IRRXUTIL("EXTRACT","USER",user,"RACF")
uid=idNum
if (RACF.OMVS.UID.1=uid) then
    say "Uid matches"
else
    say "No match"
```

- **The problem is that the REXX variable UID is overused. It is used as a variable, and also set by IRRXUTIL as part of a variable. The uses conflict. Because we cannot expect REXX programs to anticipate all possible future segment and field names, IRRXUTIL has a 'prefix' option.**

The Importance of Prefixing…

- **Let's fix the program using prefix.**

```
/* REXX */
arg user idNum
myrc=IRRXUTIL("EXTRACT","USER",user,"RACF","R_")
uid=idNum
if (RACF.R_OMVS.R_UID.1=uid) then
    say "Uid matches"
else
    say "No match"
```

- **The specified prefix is added to all variable name parts as the REXX variables are created. Specifying a prefix which you know will never be used in your program variables guarantees that there will be no name collisions.   As long as the above program does not use any variables starting with 'R_', it is safe.**

Extract Next

- **The extract next function returns the profile following the specified profile.**

- **To return the user following 'BOB', issue the following:**

```
myrc=IRRXUTIL("EXTRACTN","USER","BOB","RACF")
```

- **Repeatedly calling IRRXUTIL(EXTRACTN…) with the previously retrieved profile is a way to iterate through all profiles in a class.**

Extract NEXT for General Resource Profiles

- **When extracting General Resources with EXTRACTN, start out with non generic profiles, by specifying 'FALSE' for the GENERIC parameter.**

- **Every time IRRXUTIL(EXTRACTN…) is called, pass in the returned 'generic' indicator (stem.GENERIC), along with the returned profile name.**

- **IRRXUTIL(EXTRACTN..) will automatically switch over to GENERIC profiles when it has gone through all discrete profiles.**

Extract NEXT for General Resource Profiles

- **When extracting General Resources with EXTRACTN, start out with non generic profiles, by specifying 'FALSE' for the GENERIC parameter.**

```rexx
/* REXX */
class = 'FACILITY'
RACF.R_PROFILE = ' '
RACF.R_GENERIC= 'FALSE'
Do Forever

myrc=IRRXUTIL("EXTRACTN",class,RACF.R_PROFILE,"RACF","R_",RACF.R_GENERIC)
    If (Word(myrc,1) <> 0) Then Do
        Say myrc
        Leave
    End
    Say RACF.R_PROFILE     /* print profile name */
End
```

Specifying '.' as a Part of Stem Name

- **IRRXUTIL resets the entire supplied stem to '' (null) before populating any values. This means that each call to IRRXUTIL has new data and no residual data is left over from previous calls.**

- **If the stem variable contains a '.' (period) character, this is not possible, and IRRXUTIL does not clean anything. Return code '2' is returned as a warning that residual data has not been cleared.**

- **However, this quirk can be useful, as long as the REXX programmer is careful.**

Specifying '.' as a Part of Stem Name

- **This small program creates a small 'database' of user profile data, which is easily referenced by user id.**

```rexx
/* REXX */
arg IDS
USERS.=""                        /* only init to "", never 0 */
do i=1 to words(IDS)             /* populate specified users into USERS. stem */
  ID=word(IDS,i)                 /* Get next user                             */
  myrc=IRRXUTIL("EXTRACT","USER",ID,"USERS."||ID)
end
/* We now have all specified users saved, process them */
do i=1 to words(IDS)             /* Retrieve data from multiple users without  */
  ID=word(IDS,i)                 /*  extracting them again                      */
  say ID||" Owner="||USERS.ID.BASE.OWNER.1
end
```

```
EX 'SAMPLE(IRREXXDS)' 'ibmuser mega elvis'
 IBMUSER OWNER=IBMUSER
 MEGA Owner=SYS1
 ELVIS Owner=MEGA
 READY
```

- **A silly example, but it does illustrate extracting multiple users and indexing them nicely by user id.   By placing the user id as part of the stem, we can organize all extracted data by user id. In this example, myrc is set to '2 0 0 0 0' when successful.**

Specifying '.' as part of stem name, be careful

- **This small program shows the wrong way to use a '.' in the stem.**

```
/* REXX */
say "Extract users with no '.' in stem"
myrc=IRRXUTIL("EXTRACT","USER","MEGA","RACF","")
say "MEGA UID is "RACF.OMVS.UID.1
myrc=IRRXUTIL("EXTRACT","USER","ELVIS","RACF","")
say "ELVIS UID is "RACF.OMVS.UID.1
say "Extract users with '.' in stem to demonstrate error"
myrc=IRRXUTIL("EXTRACT","USER","MEGA","RACF.A","")
say "MEGA UID is "RACF.A.OMVS.UID.1
myrc=IRRXUTIL("EXTRACT","USER","ELVIS","RACF.A","")
say "ELVIS UID is "RACF.A.OMVS.UID.1
```

```
EX 'SAMPLE(IRREXXSR)'
  Extract users with no '.' in stem.
  MEGA UID is 8
  ELVIS UID is
  Extract users with '.' in stem to demonstrate error
  MEGA UID is 8
  ELVIS UID is
  READY
```

- **This example demonstrates how specification of a '.' in the STEM allows residual data to remain after an new extract operation.**

Where Do You Find Field Names?

- **z/OS Security Server RACF Callable Services contains tables which document every segment and field name supported by R_admin in appendix A.2**

- **Fields which are 'Returned on Extract Requests' are supported by IRRXUTIL.**

Table 107 TSO segment fields

Field name	Flag byte values	ADDUSER/ALTUSER keyword reference	Allowed on add requests	Allowed on alter requests	Returned on extract requests
ACCTNUM	'Y'	TSO(ACCTNUM (xx))	Yes	Yes	Yes
	'N'	TSO (NOACCTNUM)	No	Yes	
DEST	'Y'	TSO (DEST (xx))			
	'N'	TSO(NODEST)			

Extract?

Table 121 BASE segment fields

Field name	Flag byte values	ADDGROUP/ALTGROUP keyword reference, or LISTGROUP heading (for output-only fields)	Allowed on add requests	Allowed on alter requests	Returned on extract requests
SUPGROUP	'Y'	SUPGROUP(xx)	Yes	Yes	Yes
OWNER	'Y'	OWNER(xx)	Yes	Yes	Yes
TERMUACC (boolean)	'Y'	TERMUACC	Yes	Yes	Yes
	'N'	NOTERMUACC	Yes	Yes	
DATA	'Y'	DATA(xx)	Yes	Yes	Yes
	'N'	NODATA	No	Yes	

Segment

Field

Gotchas

- **IRRXUTIL sets the entire stem to "" (null) before setting new data. Fields which do not exist in the extracted profile remain null.**

  - This can cause problem in fields which are usually returned as numeric fields because they also remain "", and not 0. So, care must be taken before referencing numeric fields as numbers.

    ```
    /* REXX */
    arg group
    myrc=IRRXUTIL("EXTRACT","GROUP",group,"RACF","")
    do i=1 to RACF.BASE.SUBGROUP.0
       say "Subgroup: "RACF.BASE.SUBGROUP.i
    end
    ```

  The above program fails if the specified group has no SUBGROUPs because RACF.BASE.SUBGROUP.0="" which is not a number.

Gotchas…

- **Universal Groups.**

  - Remember that a universal group profile does not contain a list of the users who are connected to the group with USE authority.

- **Discrete profiles which contain generic characters will cause the underlying R_admin service to fail if they are encountered during an EXTRACTN call.**

  - IRRXUTIL fails also

  - The only solution is to RDELETE these erroneous profiles.

  - There are few cases where discrete profiles are expected to contain generic characters and R_admin handles these properly.

- **Do not beat on the RACF database. For example, do not EXTRACT-NEXT all users in an attempt to find all users which belong to a given Universal Group.**

References

- **RACF Callable Services – R_admin documentation**

- **Command Language Reference**
  - http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/Shelves/ICHZBKA0

- **Macros and Interfaces – IRRXUTIL, including an exhaustive list of all REXX variables set by IRRXUTIL.**
  - http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/ichza3a0/14.0?SHELF=EZ2ZBK0H.bks&DT=20090610215513

- **RACF Downloads page – Sample R_admin extract program (RACSEQ)**
  - http://www.ibm.com/servers/eserver/zseries/zos/racf/downloads/racseq.html

- **RACF Downloads page – IRRXUTIL examples.**
  - http://www-03.ibm.com/servers/eserver/zseries/zos/racf/downloads/irrxutil.html

IRRXUTIL Samples, from the RACF Downloads Page.

- **XDUPACL.txt** - A program which looks for user ACL entries which may be redundant with existing group ACL entries

- **XLGRES.txt** - A program which resumes the group connection of every member of a group

- **XLISTGRP.txt** - A program which displays a group's connected users in alphabetic order, with each user's name and connect authority

- **XLISTUSR.txt** - A program which displays a user's connect groups in alphabetic order

- **XRACSEQ.txt** - A program which re-implements the RACSEQ download to demonstrate features of IRRXUTIL

- **XRLIST.txt** - A program which displays the standard access list of a general resource profile with the users listed first, in alphabetic order, with the user's name, followed by the groups, in alphabetic order

- **XSETRPWD.txt** - A program which displays only the password-related SETROPTS options, and indicates whether password and password phrase enveloping is active

- **XWHOCAN.txt** - A program which displays certain users who can modify the specified profile