IBM Systems and Technology Group

IRRXUTIL: Getting RACF profile data directly into REXX programs without parsing

Mike Onghena – IBM

onghena@us.ibm.com

Disclaimer

Trademarks

Agenda

- **IRRXUTIL – REXX interface to R_admin extract.**

Overview

- **The R_admin callable service (IRRSEQ00) is an assembler programming interface which allows for management of RACF profiles and system wide settings (SETROPTS)**
- **IRRXUTIL is a load module, callable by REXX programs which calls R_admin and returns results to REXX programs.**

R_admin functions

- **Run a RACF command**
  - By providing a command image
  - By providing tokenized data
- **Extract user, group, general resource profile information and SETROPTS settings. This is the only function supported by IRRXUTIL.**
- Retrieve a PKCS#7 password envelope

What is IRRXUTIL?

- IRRXUTIL is a load module, shipped in **z/OS V1R11** which is called by REXX programs to extract RACF profile data.
- IRRXUTIL calls the R_admin extract functions to extract USER, GROUP, CONNECT, RESOURCE and SETROPTS data from RACF.
- The resulting profile data is then injected directly into REXX variables.
- On successful return from IRRXUTIL, RACF profile data is ready to use, just by referencing REXX variables.

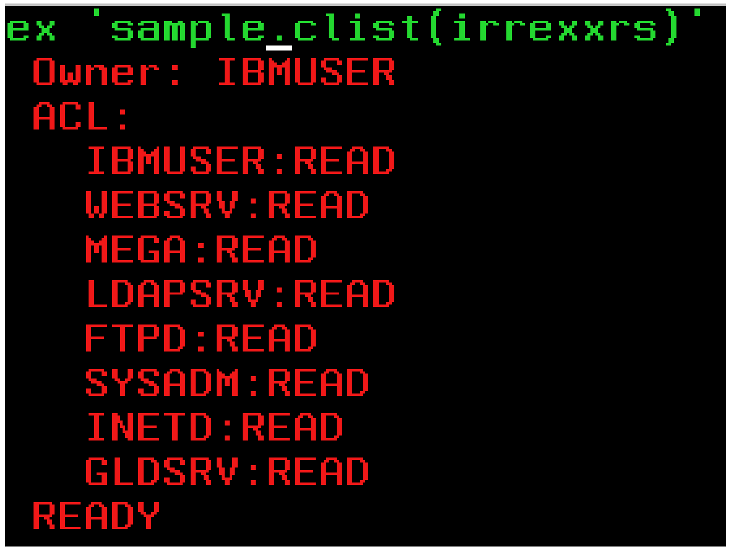
What IRRXUTIL is not

- IRRXUTIL does not have any support for any of the other function codes supported by R_admin.
- 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.

Super Simple Silly Sample

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

```
/* 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
```



```
ex 'sample.clist(irrexxrs)'
 Owner: IBMUSER
 ACL:
   IBMUSER:READ
   WEBSRV:READ
   MEGA:READ
   LDAPSRV:READ
   FTPD:READ
   SYSADM:READ
   INETD:READ
   GLDSRV:READ
 READY
```

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

What's the catch?

- The caller does need access to use R_admin extract via the appropriate FACILITY class profile protecting the desired function.
- The caller must be allowed to retrieve the profile in question.
- The caller will only have fields they are allowed to view returned.
- R_admin will enforce all field-level-access-checking rules.
- 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

How does it work?

- myrc=**IRRXUTIL(*function,type,profile,stem,prefix,generic*)**
  - ***Function*** - “EXTRACT” or “EXTRACTN”
  - ***Type*** – “USER”, “GROUP”, “CONNECT”, “_SETROPTS”, ***any*** *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.

IRRXUTIL return code

- **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**
- **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 */
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
```

2 ways to process IRRXUTIL results

- **The variables which are set by IRRXUTIL can be used in 2 ways, depending on the application**
  - 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.
  - Sadly, there is no mechanism to find out all *potential* segments/field which could exist. It only returns what exists for a given profile.

Direct retrieval of data by segment and field

- **Stem variables have the form:**
  - stem.segment-name.field-name.0 = number of values
  - stem.segment-name.field-name.n = nth value of field
- **For a simple non-repeating field:**
  - stem.segment-name.field-name.0 = 1
  - stem.segment-name.field-name.1 = value
- **A repeating field may have more than 1 value:**
  - stem.segment-name.field-name.0 = 2
  - stem.segment-name.field-name.1 = value1
  - stem.segment-name.field-name.2 = value2
- **Examples (where stem = RACF)**
  - RACF.BASE.SPECIAL.0 = 1
  - RACF.BASE.SPECIAL.1 = TRUE
  - RACF.OMVS.UID.0 = 1
  - RACF.OMVS.UID.1 = 555

Additional control information for fields

Name	Description	Example
stem.CLASS	Class Name	PROF.CLASS = “USER"
stem.PROFILE	Profile Name	PROF.PROFILE=”IBMUSER”
stem.GENERIC	TRUE or FALSE	PROF.GENERIC=”FALSE”

stem.segname. fieldname.OUTPUTONLY	TRUE or FALSE	PROF.BASE.CREATDAT.OUTPUTONLY=”TRUE”
		
		PROF.BASE.SPECIAL.OUTPUTONLY=”FALSE”

stem.segname. fieldname.BOOLEAN	TRUE or FALSE	PROF.BASE.SPECIAL.BOOLEAN=”TRUE”
		
		PROF.BASE.NAME.BOOLEAN=”FALSE”

stem.segname. fieldname.REPEATING	TRUE or FALSE – Does this field have more than 1 value?	PROF.BASE.UAUDIT.REPEATING=”FALSE”
		
		PROF.BASE.CGROUP.REPEATING=”TRUE”

stem.segname. fieldname.REPEATCOUNT	Number of occurrences of repeat group.
	
	Repeat header field only.
		PROF.BASE.CONNECTS.REPEATCOUNT=5
		
		PROF.BASE.SPECIAL.REPEATCOUNT=0

A complete table appears in the Macros and Interfaces Book.

Retrieving unknown data

A number of variables are set which define which segments and fields have been retrieved.

- Stem.0 = number of segments
- Stem.1-n = names of segments
- Stem.*segment*.0 = Number of fields in a segment
- Stem.*segment*.1-n = Field names in that segment
- Stem.*segment.field*.0 = # values for field
- Stem.*segment.field*.0 = Field values

Much needed example on next page

Retrieving unknown data example

```
stem.0 = 4
    .1 = “BASE”
    .2 = “TSO”
    .3 = “OMVS”
    .4 = “CICS”

    .PROFILE = “BRWELLS”
    .CLASS   = “USER”
    .GENERIC = “FALSE”
    .VERSION = 0
```

```
stem.BASE
.0 = 30
.1 = “NAME”
.2 = “SPECIAL”
.3 = “CLAUTH”
.4 = … … …

.FLAGS = “00000000”
```

```
stem.BASE.NAME
.0 = 1
.1 = “BRUCE WELLS”
```

```
stem.BASE.SPECIAL
.0 = 1
.1 = “FALSE”
```

```
stem.BASE.CLAUTH
.0 = 3
.1 = “USER”
.2 = “FACILITY”
.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 – simple repeating field

```
 stem.BASE

.0 = 30
.1 = “NAME”
.2 = “SPECIAL”
.3 = “CLCNT”
.4 = “CLAUTH”
.5 = “CONNECTS”
.6 = “CGROUP”
.7 = “CAUTHDA”
.8 = “COWNER”
.n = … … …
```

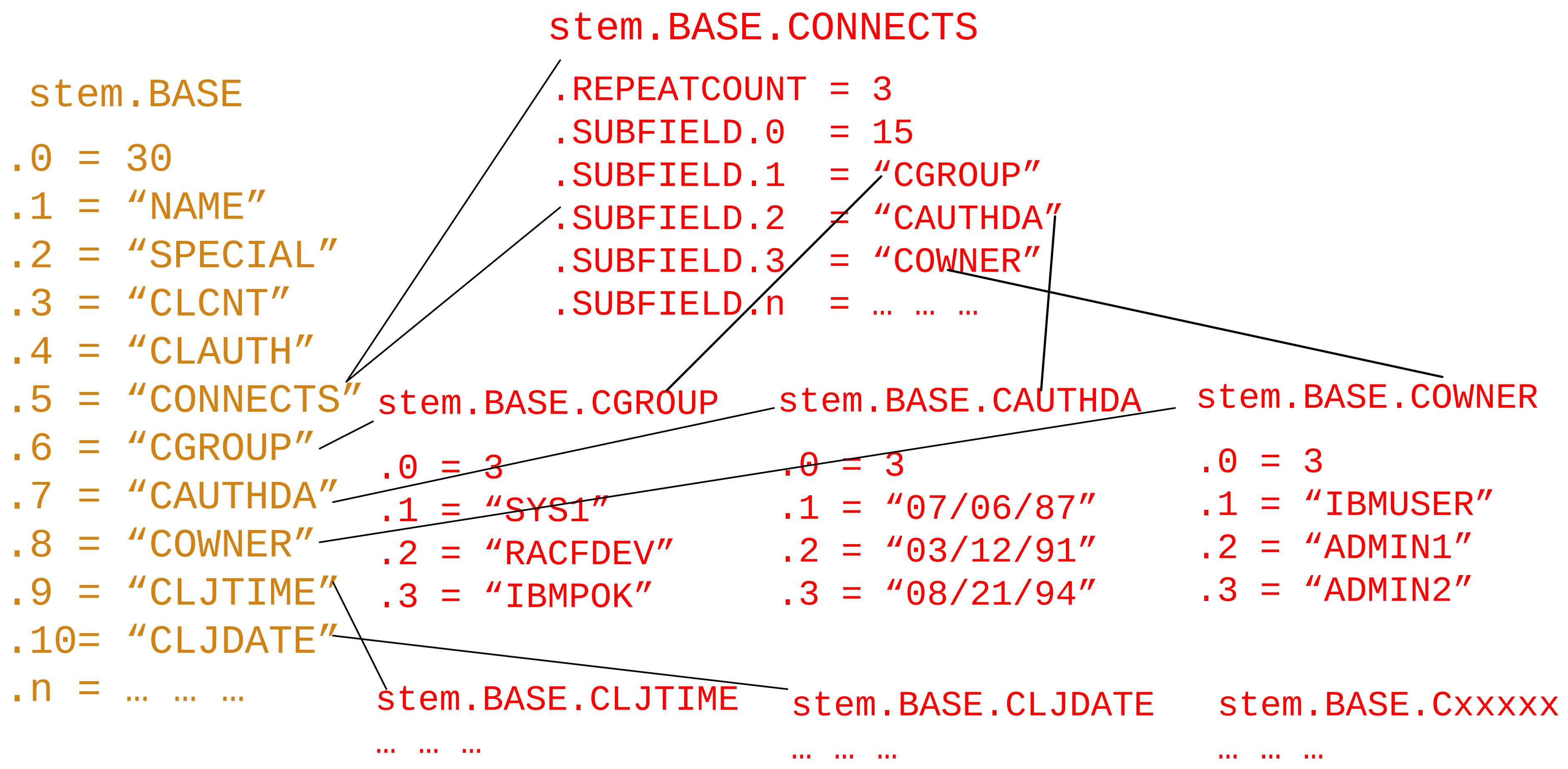
```
stem.BASE.CLCNT

.REPEATCOUNT = 3
.SUBFIELD.0  = 1
.SUBFIELD.1  = “CLAUTH”
.REPEATING   = “FALSE”
.OUTPUTONLY = “TRUE”
.BOOLEAN     = “FALSE”
```

```
stem.BASE.CLAUTH

.0 = 3
.1 = “USER”
.2 = “FACILITY”
.3 = “UNIXPRIV”
.REPEATING   = “TRUE”
.OUTPUTONLY = “FALSE”
.BOOLEAN     = “FALSE”
```

Stem structure – complex repeating field



```
stem.BASE

.0 = 30
.1 = "NAME"
.2 = "SPECIAL"
.3 = "CLCNT"
.4 = "CLAUTH"
.5 = "CONNECTS"
.6 = "CGROUP"
.7 = "CAUTHDA"
.8 = "COWNER"
.9 = "CLJTIME"
.10= "CLJDATE"
.n = … … …
```

```
stem.BASE.CONNECTS
.REPEATCOUNT  = 3
.SUBFIELD.0   = 15
.SUBFIELD.1   = "CGROUP"
.SUBFIELD.2   = "CAUTHDA"
.SUBFIELD.3   = "COWNER"
.SUBFIELD.n   = … … …
```

```
stem.BASE.CGROUP        stem.BASE.CAUTHDA        stem.BASE.COWNER

.0 = 3                  .0 = 3                   .0 = 3
.1 = "SYS1"             .1 = "07/06/87"          .1 = "IBMUSER"
.2 = "RACFDEV"          .2 = "03/12/91"          .2 = "ADMIN1"
.3 = "IBMPOK"           .3 = "08/21/94"          .3 = "ADMIN2"

stem.BASE.CLJTIME       stem.BASE.CLJDATE        stem.BASE.Cxxxxx
… … …                   … … …                    … … …
```

Prefix, why it is important

- **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 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.

Prefix, why it is important

- **Lets 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 resources

- **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 resources

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

```
/* 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
```

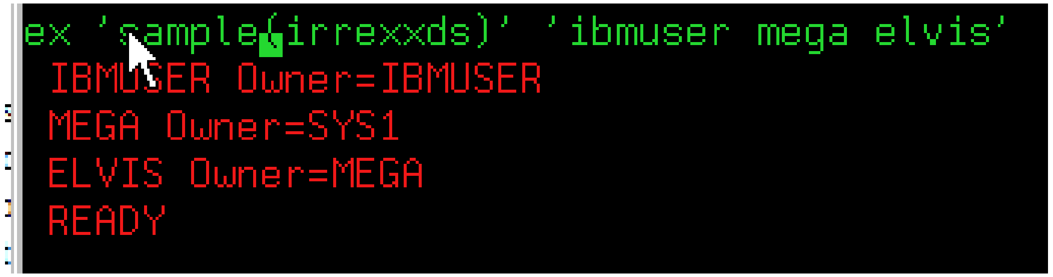
Specifying '.' as part of stem name

- **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 part of stem name

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

```
/* 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
```



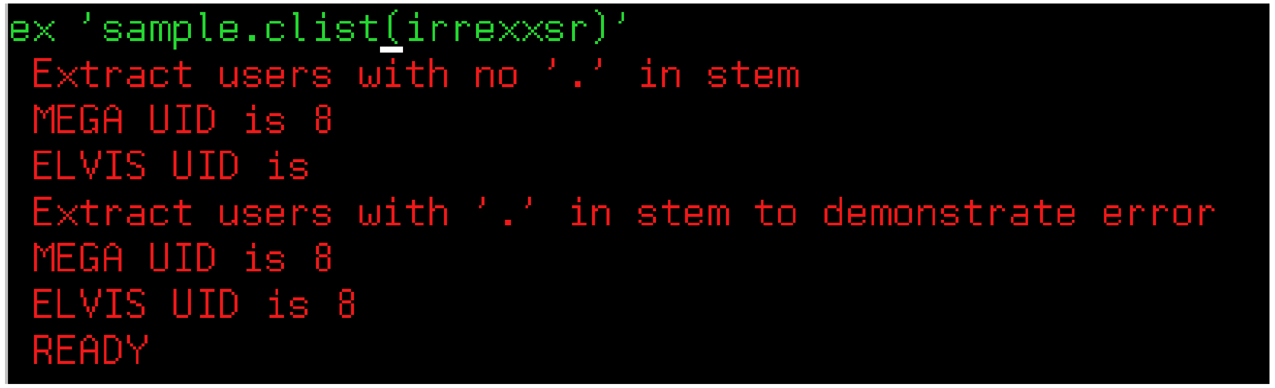
```
ex 'sample(irrexxds)' 'ibmuser mega elvis'
 IBMUSER Owner=IBMUSER
 MEGA Owner=SYS1
 ELVIS Owner=MEGA
 READY
```

- 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.clist(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 8
 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.**

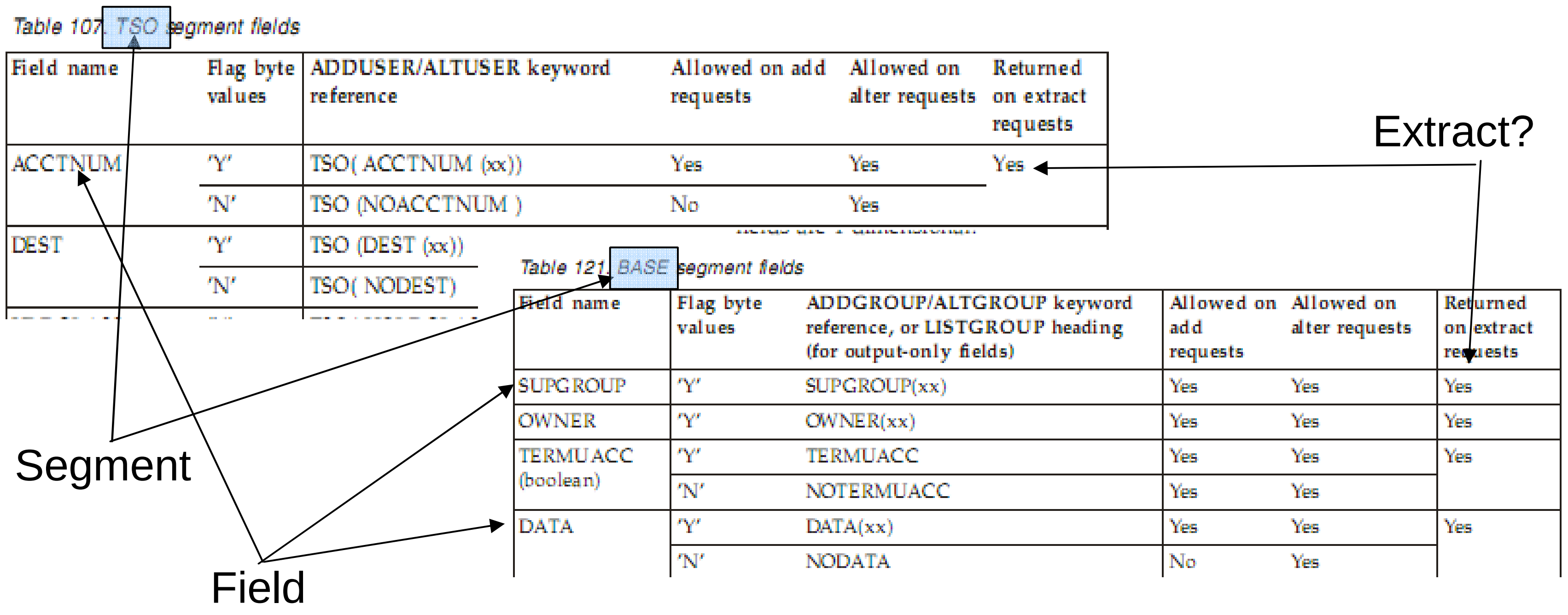

Extract?
Segment
Field

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)			

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	

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.

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

Gotchas

- Universal Groups. Although not a direct issue with IRRXUTIL, the presence of universal groups makes certain types of problem much harder to solve. A Universal group is a RACF group which does not contain a list of the user ids connected to the group. The connection information is stored solely in the user id profiles. This makes for more efficient operation, but any program which relies on running the list of users in the group will not work for Universal groups.
- 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-03.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 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**