



IBM Systems and Technology Group

# IRRXUTIL – REXX interface for retrieving RACF profile data

**RUGONE**

**October 17, 2013**

**Bruce R. Wells – IBM**

**[brwells@us.ibm.com](mailto:brwells@us.ibm.com)**

# Disclaimer

**The information contained in this document is distributed on an "as is" basis, without any warranty either express or implied. The customer is responsible for use of this information and/or implementation of any techniques mentioned. IBM has reviewed the information for accuracy, but there is no guarantee that a customer using the information or techniques will obtain the same or similar results in its own operational environment.**

**In this document, any references made to an IBM licensed program are not intended to state or imply that only IBM's licensed program may be used. Functionally equivalent programs that do not infringe IBM's intellectual property rights may be used instead. Any performance data contained in this document was determined in a controlled environment and therefore, the results which may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.**

**It is possible that this material may contain references to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM Products, programming or services in your country.**

**IBM retains the title to the copyright in this paper as well as title to the copyright in all underlying works. IBM retains the right to make derivative works and to republish and distribute this paper to whomever it chooses.**

# Trademarks

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

- z/OS
- RACF

\* Registered trademarks of IBM Corporation

**The following are trademarks or registered trademarks of other companies.**

Java and all Java-related trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.

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

SET and Secure Electronic Transaction are trademarks owned by SET Secure Electronic Transaction LLC.

\* All other products may be trademarks or registered trademarks of their respective companies.

## Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

# Agenda

- **R\_admin profile-extract functions**
- **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)**
- **Easier to use than RACROUTE or ICHEINTY**
- **User/group extract functions added and documentation completely rewritten in z/OS V1R7**
- **General resource extract added in z/OS V1R11**

## R\_admin functions

- **Run a RACF command**
  - By providing a command image
  - By providing tokenized data
- **Extract user, group or general resource profile information**
- **Extract SETROPTS settings**
- **Retrieve a PKCS#7 password envelope**

## R\_admin as a SAF Interface

- **R\_admin called by SAF router, subject to SAF exits**
- **But it is a highly RACF-specific interface**
  - Segment names, field names, data format
- **Don't expect this to be a general administrative interface which will work regardless of the underlying security product**

## Call parameters

```
CALL IRRSEQ00, (Work_area,          /* Common parms */
               ALET, SAF_return_code, /* for all the */
               ALET, RACF_return_code, /* RACF callable */
               ALET, RACF_reason_code, /* services */
               Function_code,        /* Requested fcn */
               Parm_list,            /* Input p-list */
               RACF_userID,          /* "Run-as" user */
               ACEE_ptr,              /* "Run-as" ACEE */
               Out_message_subpool,  /* Output subpool */
               Out_message_strings   /* Output anchor */
               ), VL
```



## R\_admin General Attributes

- **Caller specifies the function to perform and provides a function-specific parameter list**
- **Caller provides a subpool and address field for the output**
- **Supervisor state callers can specify an identity under whose authority the request will run**
- **Some functions are available to problem state callers, and are protected by FACILITY resources**
- **Many functions require the RACF subsystem address space. Caller does not require a TSO environment to issue a command.**

Note: IRRPCOMP macro provides some mappings and constants

# Profile Extract Functions



## Profile extract functions

- **Extract User, Group, Connect and General Resource information from the RACF database in an architected format which is a programming interface**
- **No limit imposed on output size**
- **Requires same authority as “list” cmd processor**
- **All (authorized) profile data returned**
- **Dataset not supported**

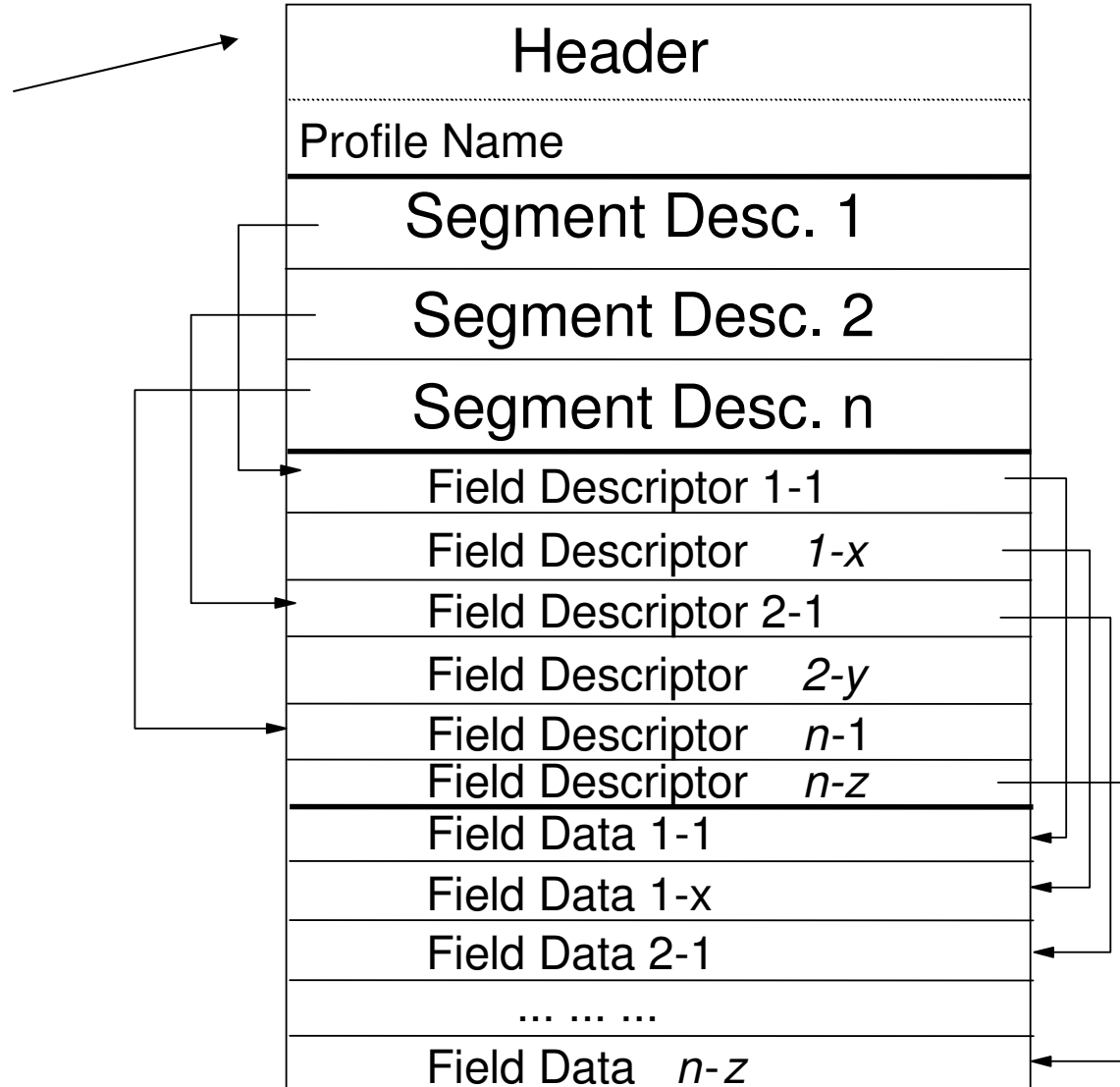
Function codes	Authorization	RACF address space required
25-29,31-32	<ul style="list-style-type: none"><li>■ Command processor authorization</li><li>■ FACILITY - IRR.RADMIN.&lt;cmd-name&gt; (READ)</li></ul>	No

## R\_admin extract as a hybrid of a command processor and RACROUTE REQUEST=EXTRACT

Like RACROUTE	Like a command
Format is architected (i.e. Easily read by program)	Returned data is character (EBCDIC)
Supervisor state caller can bypass authorization	Returned data is in the same format as accepted by commands, eg, dates, numbers
Runs in caller's address space (much faster than run-command)	Problem state enabled – requires same authorization as command
Can iteratively cycle through profiles	Suppresses fields not displayed by the command processor

# Profile extract output format

Parm\_list (input)  
and  
Out\_message\_strings  
(output)



## Repeat Fields (aka “multi-value” fields)

- **N-dimensional repeating data fields. E.G.**
  - Class authority (CLAUTH) – 1-dimensional
  - Group connection in user profile – 15-dimensional
- **Header field descriptor with unique name identifies**
  - Number of occurrences of repeat field
  - Number of elements (dimension) in field
- **Subsequent field descriptors for each constituent field, repeated as necessary**

## “Next” requests

- **For users, groups and general resources (not connections), you can iterate through the profiles by providing a starting value for profile name**
  - Next name is returned, similar to ICHEINTY NEXT or RACROUTE REQUEST=EXTRACT TYPE=EXTRACTN
- **The output of the  $n$ th request can be used as the input of the  $n+1$ th request**
  - You need only re-specify flags, if desired

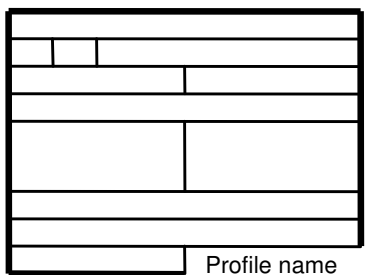
## “Next” processing

- 1. Build the plist header. Specify a profile name of a single blank to start at the top.**
- 2. Call IRRSEQ00 passing the plist in the Parm\_list parameter. Output returned in Out\_message\_strings parameter.**
- 3. Free original (or  $n-1$ ) plist.**
- 4. Process the output as appropriate.**
- 5. (Re)set header flags, as appropriate.**
- 6. Call IRRSEQ00 with  $n-1$  output as  $n$  input.**
- 7. Iterate at step 3 until finished (RC 4/4/4).**



# “Next” Processing (with pictures)

Build Plist header

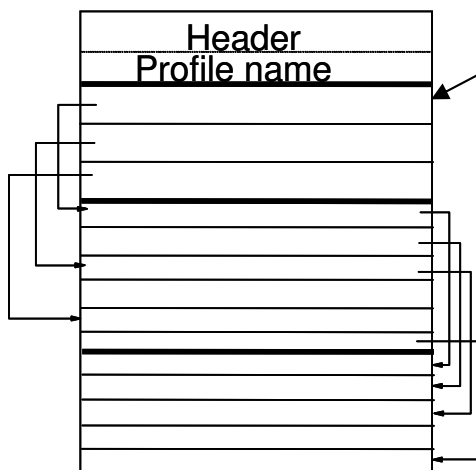


Call R\_admin

```
CALL IRRSEQ00, (Work_area,
                ALET, SAF_return_code,
                ALET, RACF_return_code,
                ALET, RACF_reason_code,
                Function_code,
                Parm_list,
                RACF_userID,
                ACEE_ptr,
                Out_message_subpool,
                Out_message_strings
                ), VL
```

Free previous storage

Process output



Until done (SAF RC4, RACF RC4, RACF reason code 4 means no more profiles)

## NEXT processing for Resources

- **When iterating through general resources, all discrete profiles are first, followed by all generic profiles.**
- **To see all of them, start with x'40' (blank) as the profile name and turn the 'generic' flag off.**
- **After the last discrete profile is returned, R\_admin will automatically switch to generic profiles and set the 'generic' flag in the header accordingly.**
- **Subsequent calls to extract-next will return generic profiles.**
- **After the last generic profile is returned, r\_admin will return 'not found'.**
- **When 'respecifying' the flags, as recommended on an earlier slide, be sure to NOT reset this generic flag.**

# SETROPTS Reporting Functions



## SETROPTS reporting functions

- **Retrieve SETROPTS settings in one of two formats**
  - SMF Unload (Type 81)
  - SETROPTS input format (tokenized)
    - Not the same as R7 extract format (Sorry!)
- **Very simple: no input parameter list required**
- **Have been around “forever”, but “extract” was problem-state enabled in z/OS R11 (for IRRXUTIL)**

Function codes	Authorization	RACF address space required
22, 23	<ul style="list-style-type: none"> <li>■ Unload, Extract pre-R11:               <ul style="list-style-type: none"> <li>■ Supervisor state</li> <li>■ SETROPTS LIST authority *not* checked</li> </ul> </li> <li>■ R11 Extract:               <ul style="list-style-type: none"> <li>■ FACILITY - IRR.RADMIN.SETROPTS.LIST (read)</li> <li>■ Command processor (SETROPTS LIST) authorization</li> </ul> </li> </ul>	No

## IRRXUTIL – REXX interface to R\_admin extract functions



## 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 image and run it directly from REXX.

## Simple example

- 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 "Profile name: "||RACF.profile  
do a=1 to RACF.BASE.ACLCNT.REPEATCOUNT  
  Say "  "||RACF.BASE.ACLID.a||": "||RACF.BASE.ACLACS.a  
end
```

```
ex 'sample_clist(irrexrs)'  
Profile name: BPX.DAEMON  
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 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*. DATASET not supported.
  - **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	
<a href="#">R_admin failure</a>	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 (or 2), the call was successful.

```
/* REXX */  
myrc=IRRXUTIL("EXTRACT","FACILITY","BPX.DAEMON","RACF","","FALSE")  
If (word(myrc,1)>2) 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
<b>stem.CLASS</b>	<b>Class Name</b>	<b>PROF.CLASS = "USER"</b>
<b>stem.PROFILE</b>	<b>Profile Name</b>	<b>PROF.PROFILE="IBMUSER"</b>
<b>stem.GENERIC</b>	<b>TRUE or FALSE</b>	<b>PROF.GENERIC="FALSE"</b>
<b>stem.segname. fieldname.OUTPUTONLY</b>	<b>TRUE or FALSE</b>	<b>PROF.BASE.CREATDAT.OUTPUTONLY="TRUE"</b> <b>PROF.BASE.SPECIAL.OUTPUTONLY="FALSE"</b>
<b>stem.segname. fieldname.BOOLEAN</b>	<b>TRUE or FALSE</b>	<b>PROF.BASE.SPECIAL.BOOLEAN="TRUE"</b> <b>PROF.BASE.NAME.BOOLEAN="FALSE"</b>
<b>stem.segname. fieldname.REPEATING</b>	<b>TRUE or FALSE</b> – Does this field have more than 1 value?	<b>PROF.BASE.UAUDIT.REPEATING="FALSE"</b> <b>PROF.BASE.CGROUP.REPEATING="TRUE"</b>
<b>stem.segname. fieldname.REPEATCOUNT</b>	<b>Number of occurrences of repeat group.</b>  <b>Repeat header field only.</b>	<b>PROF.BASE.CONNECTS.REPEATCOUNT=5</b> <b>PROF.BASE.SPECIAL.REPEATCOUNT=0</b>

A complete table appears in the Macros and Interfaces Book.



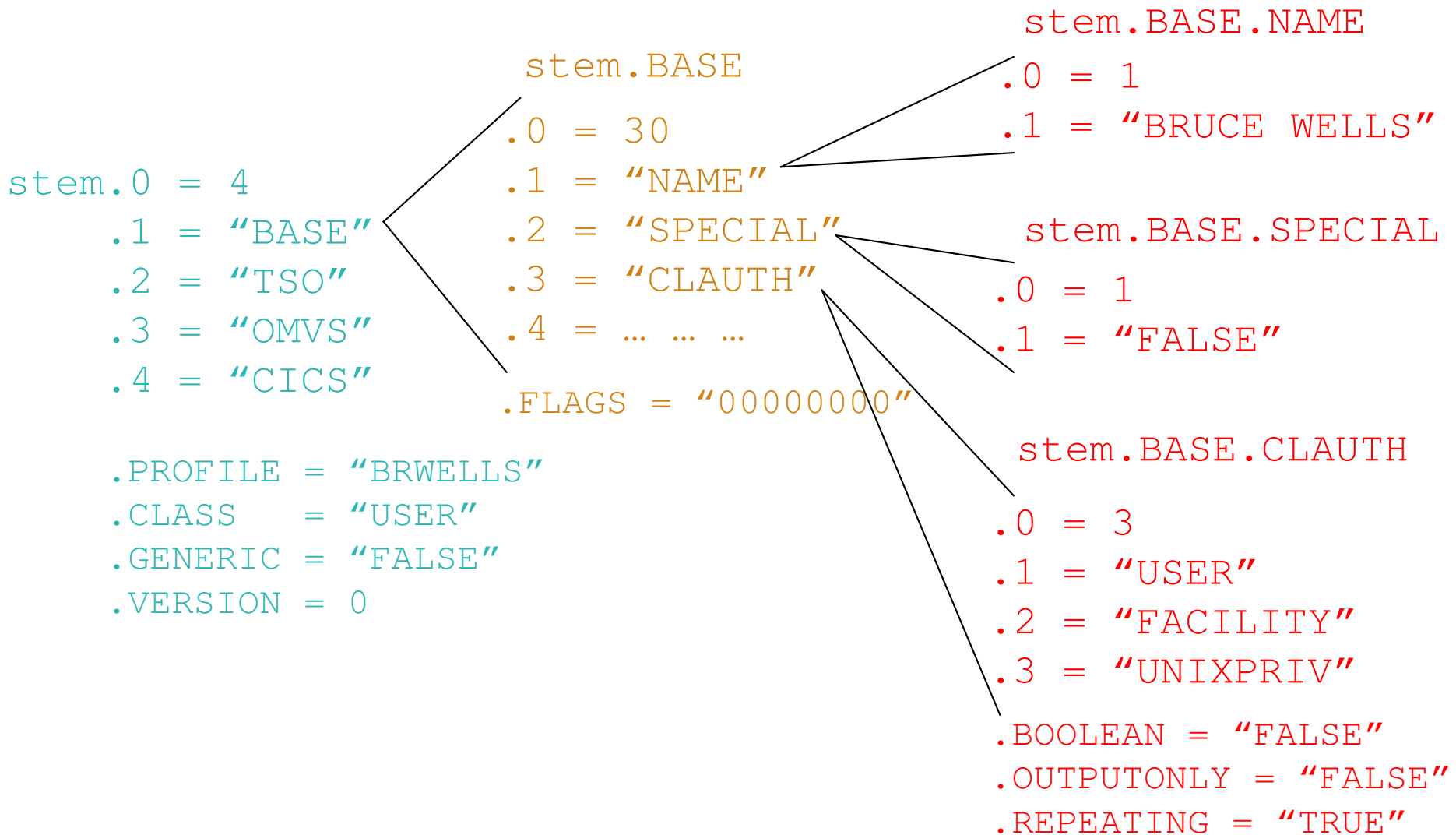
## Retrieving unknown data

**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



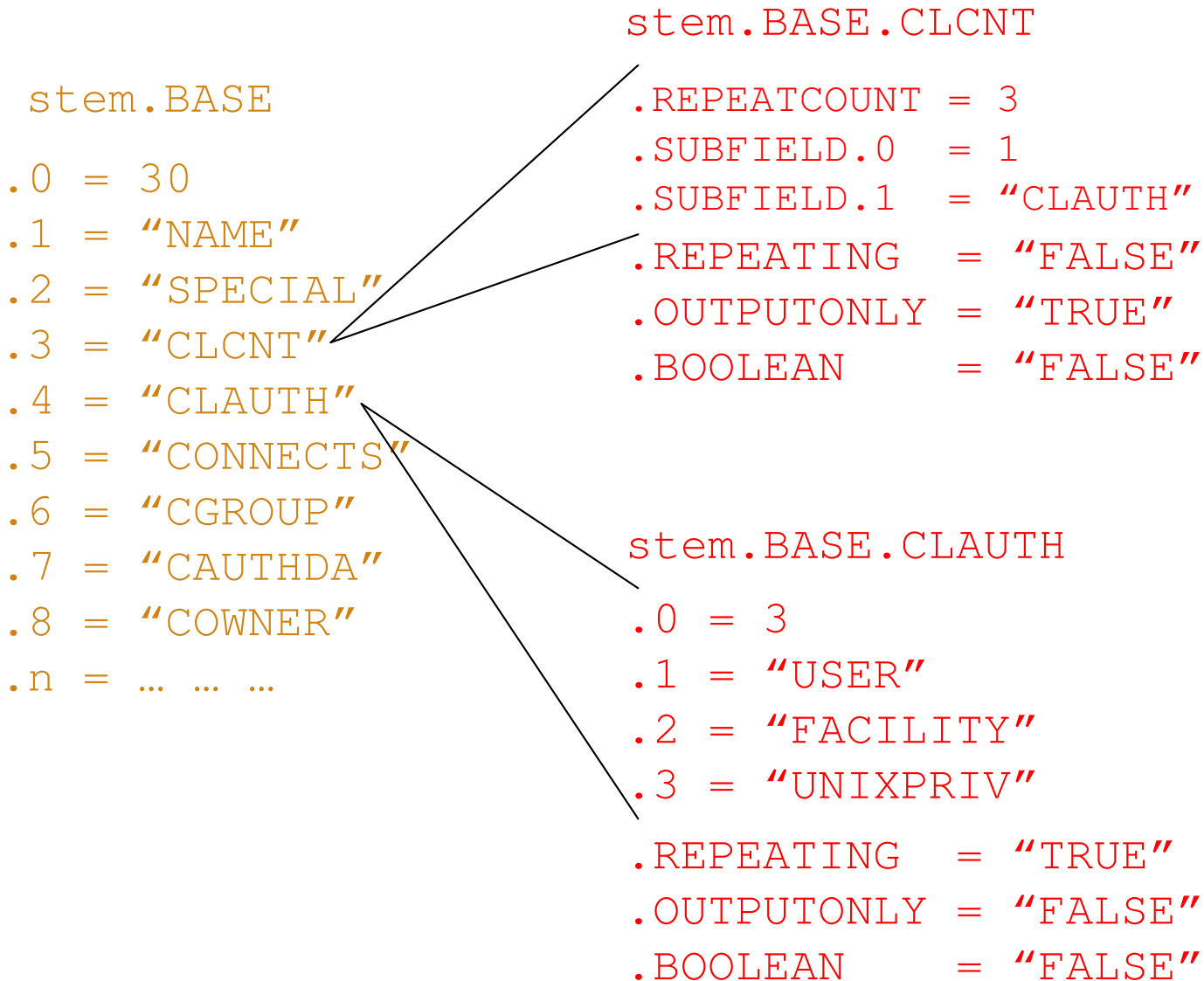
## Retrieving repeating data

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

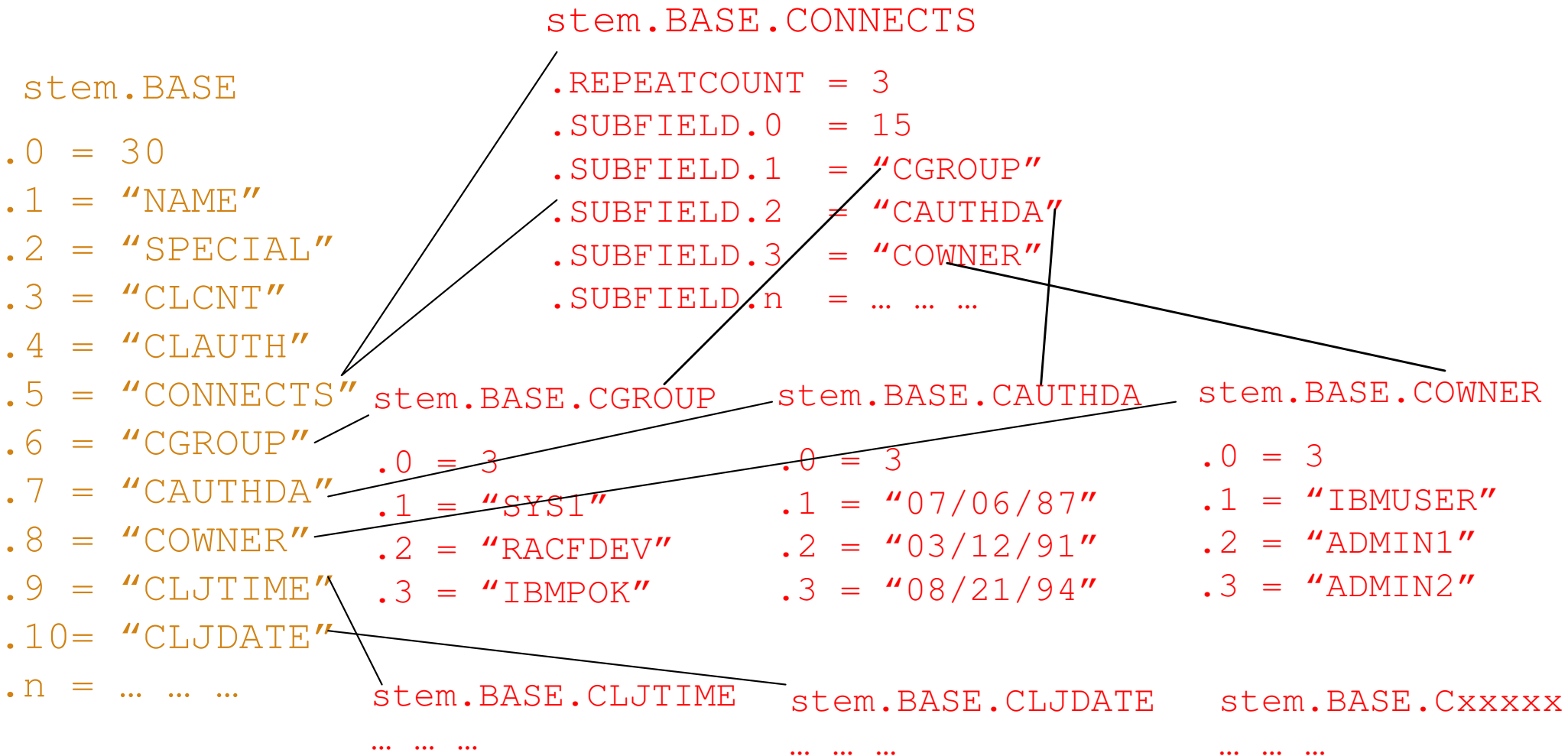
- `Stem.segment.field.repeatCount`. Non-zero value indicates *field* is a repeat header. This is the number of repeat groups for this field.
- `Stem.segment.field.subfield.0` = Number of subfields in this repeat group.
- `Stem.segment.field.subfield.1-n` = subfield names
- `Stem.segment.subfieldname.0` = 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 structure – complex repeating field



## 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=IRRUTIL("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(irrexds)' '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.**

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	

Segment

Field

Extract?

## 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.**

## References

- [RACF Callable Services – R\\_admin documentation](#)
  - [Field tables](#)
- [Command Language Reference](#)
- [Macros and Interfaces – IRRXUTIL](#), including an exhaustive [list](#) of all REXX variables set by IRRXUTIL.
- [RACF Downloads page](#)
  - IRRXUTIL [examples](#).
  - [RACSEQ](#) – sample R\_admin program