IRRXUTIL Sample Programs

z/OS® Security Server RACF®

Authors: Michael Onghena and Bruce R. Wells

brwells@us.ibm.com

onghena@us.ibm.com

IBM® Corporation

Last updated: 10/25/2022

Change Date	Change Description
01/13/2010	Introduction of samples
10/16/2014	Addition of XSTARTED and update to XSETRPWD for OA43999
10/25/2022	• Addition of XCDT, XCDTSRCH, XCDTSUM, XGRPALU, XRDEFROM, XRRSFPFX, XSETRAUD • Moved download to GitHub

1 Introduction

In z/OS V1R11 a REXX interface called IRRXUTIL was officially introduced in RACF. It acts as a front-end to the R_admin callable service (IRRSEQ00) for the purpose of extracting profile and SETROPTS information and places the information into a REXX stem for consumption by the caller.

IRRXUTIL is fully documented in RACF Macros and Interfaces, and we do not intend to repeat that documentation here. Having said that, we will provide a 20,000 foot view of the interface below.

2 Package Contents

This package contains

- This README file
- X*name*.txt – the individual samples (documented below) in text format for casual viewing
- IRRXUTIL.xmit – All of the sample programs in TSO XMIT format, for your convenience to get these samples into a PDS on z/OS.

3 IRRXUTIL Overview

There are essentially three things you need to know about IRRXUTIL:

- how to invoke it
- how to interpret the return codes
- the variable names that are set

3.1 How to invoke it

IRRXUTIL is invoked from a REXX program as follows:

```
myrc=IRRXUTIL(command,type,profile,stem,prefix,generic)
```

You tell IRRXUTIL what function (*command*) to perform (extract or extract-next), the *type* (or class) of profile to extract, the *profile* name, and the name of a *stem* variable (without the trailing period) in which to return the information. The final two parameters are optional: a prefix to apply to the generated variable names in order to reduce collisions with existing variable names in a REXX program, and a generic flag that tells RACF to extract from a matching generic general resource profile if a discrete profile does not exist.

3.2 How to interpret the return codes

There is not a single return code, but a string of five return codes separated by a blank. These are documented in Macros and Interfaces, and there isn't much sense repeating them here. For the most part, you will only need to know whether the first value ("return code 1" in the documentation) is zero or not. For example,

```
if (word(myrc,1)=0) then …
```

If it is not zero, then the remaining values will help you determine exactly what the error is.

Keep in mind the special value of 2 for "return code 1". This also means that everything was OK, but you specified a stem value which contained a period. If you did not mean to do this, then the value of 2 serves as a warning. If you meant to do this, you should treat a value of 2 as successful.

3.3 The variable names that are set

Most often, you will probably extract a profile knowing exactly which field(s) you want. In this case, the fields can be access directly once you understand the naming convention.

- *stem.segment.field*.0 contains the number of values for the field. For a normal field, this value will be 1. It will only be greater for repeating data.
- *stem.segment.field.n* contains the *n*th value for the field. For most fields, *n* will be 1.

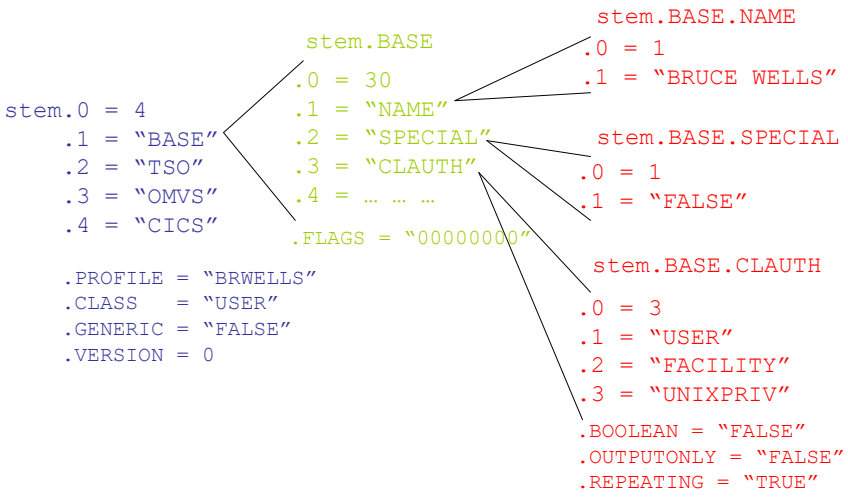
If you specify a prefix, this naming will be altered somewhat.

There is a whole lot more information returned, however. The stem variables returned are designed in such a way that, knowing nothing about the contents of a given RACF profile, all the defined segments, the fields they contain, and the values of those fields can be systematically accessed. This could be helpful when simply dumping out the contents of a profile to the screen, or populating a table.

stem.0 contains the number of segments returned. *stem.n* contains the name of the *n*th segment. Using REXX substitution, you can then access *stem.segment*.0 which contains the number of fields. *stem.segment.n* contains the name of the *n*th field. You can then access *stem.segment.field*.0 which contains the number of values for the field. *stem.segment.field.n* contains the *n*th value of the field. For repeating fields, you can determine the names of the subfields, and how many instances of the field are present.

But wait, there's more! There is a set of stem variables at the profile, segment, and field level which provide additional information.

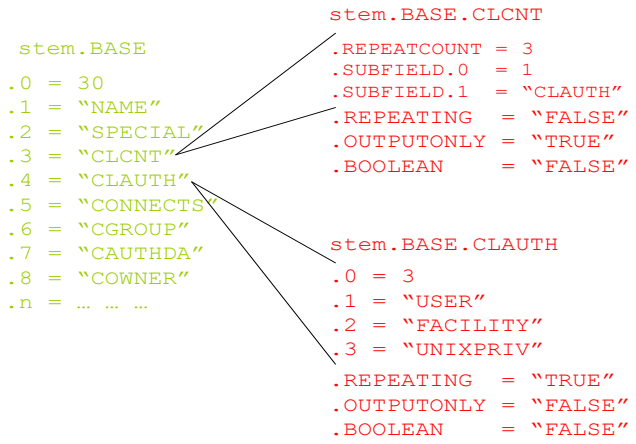
The following picture shows how the variable names are organized.



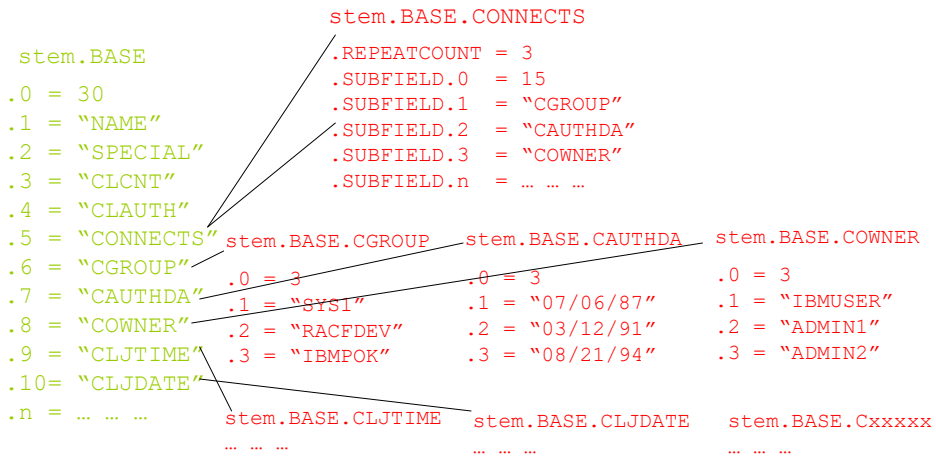
Repeating fields have a "summary" field (called a "header" field in R_admin documentation) which contains stem variables that indicate the number of occurrences of the field, and the names of the subfields. The subfield names are enumerated in a similar fashion as the other stem variables we have already seen. *stem.segment.field*.SUBFIELD.0 contains the number of subfields, and *stem.segment.field*.SUBFIELD.*n* contains the name of the *n*th subfield. *stem.segment.field*.REPEATCOUNT contains the number of occurrences of the repeating field.

And the “summary” field and each subfield have the same set of field-level attributes we have already seen. The interface may seem a bit cluttered here as the attribute fields are defined for every field even if they aren’t relevant. For example, the summary field has a BOOLEAN indicator which will always be FALSE. But no matter, just pretend these fields don’t exist if you don’t care about them.

Here’s an example of a simple repeating field: a user’s class authority (CLAUTH). It has a header field named CLCNT, and then a single subfield named CLAUTH which contains all the values.



The repeat field stuff makes more sense (as to why it's needed) when you consider a complex field like an access list or a list of group connections. In the following example, a user's group connections are shown. The header field named CONNECTS is associated with no less than 15 subfields (not all of which are shown).



And that's it in a nutshell! (So maybe the nut is on the large side, like a coconut perhaps.) You will get the hang of it pretty quickly.

4 Installation Instructions

1. Download the source code (IRRXUTIL.xmit) to your workstation using a browser. Then transfer it to your z/OS system using FTP in binary mode. You must transfer it into a fixed-block 80 data set. For example, on Windows, this can be accomplished by specifying the following FTP client command before initiating the transfer

   quote site lrecl=80 recfm=fb blksize=0

2. From a TSO session on z/OS, issue the RECEIVE command to unpackage the file. The syntax of the RECEIVE command is:

   RECEIVE INDATASET(dsname)

   RECEIVE prompts you for a target data set name.

   Note: If you receive a message from the RECEIVE command that indicates the input data set is in an incorrect format, verify that:
   - The files were FTP'd in binary format
   - The input files are in fixed block format

3. Inspect the sample code, and modify it if desired.

4. Run it!

5 The Samples

In the interests of brevity and clarity, the samples do not include full parameter validation and error checking. Their intent is to demonstrate various aspects of IRRXUTIL and not intended to be used out of the box as ready to deploy applications. If you have questions or concerns about the samples, please post to racf-l. Please do not contact IBM support.

5.1 XCDT – Display all attributes of a RACF class

Uses the extract-CDT function of IRRXUTIL to display every attribute used to define a RACF class, static (using the ICHERCDE macro) or dynamic (using the CDTINFO segment of CDT class profiles). Fr static classes, this sure beats looking it up in Macros and Interfaces, and guessing at what the default values are for attributes that aren't listed!

XCDT can also use SETROPTS-extract to display the operational (SETROPTS) settings in effect for the class. This sure beats crawling through SETROPTS LIST output!

The reported data is 'live' on the system, so there is no guesswork as to when the last SETROPTS RACLIST(CDT) REFRESH command was issued.

https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/IRRXUTIL/XCDT.txt

5.2 XCDTSRCH – Find all RACF classes with a certain attribute

Uses the extract-CDT function of IRRXUTI to find every RACF class with a certain attribute. Have you ever wanted to know which classes issue an ENF signal? Or which support mixed case profiles? Just use the XCDT exec to display an arbitrary class. That output shows all the REXX variable names in parentheses. Look for the attribute in which you are interested. This is the attribute you will specify on XCDTSRCH.

https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/IRRXUTIL/XCDTSRCH.txt

5.3 XCDTSUM – Find all RACF classes with a certain attribute

Uses the extract-CDT function of IRRXUTI, and ISPF tables, Display summary of RACF classes and list by ascending POSIT number. XCDTSUM can be used to confirm a POSIT number is available for use.

https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/IRRXUTIL/XCDTSUM.txt

~~5.1~~5.4 XDUPACL – Search for duplicate USER entries in Access Control Lists

Given a RACF class name and a number of profiles, search these profiles for possibly redundant USER entries. That is, search the access list and identify all user ids which are granted access

because the user id is explicitly on the access list, and is again on the access list by virtue of membership in a group which is also present on the resource access list.

The sample demonstrates the use of 'period in a stem' processing to build and organize a list of user id profiles in a single REXX stem variable which can be easily accessed repeatedly. This is done in the 'indexID' subroutine. We save the user ids to reduce read operations to the RACF database in the event that we check more than one profile.

This sample is limited for simplicity, and to reduce the amount of irrelevant parsing code. It could be easily expanded to handle more than one class, possibly by specifying the list of resources in a CLASS(resource [resource]..) or CLASS.RESOURCE format.

Another enhancement would be to accept a list of user ids to check. If such a list of user ids was included (again, just a matter of more parsing and a little tweaking of the code), XDUPACL could check to see if there were any redundancies based solely on group membership. Without direct specification of the user ids, this is not possible due to the existence of universal groups.

The sample only processes the standard access list and NOT the conditional access list. Again, this could be added if desired.

https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/IRRXUTIL/XDUPACL.txt

5.5 XGRPALU – Issue ALTUSER against every member of a group

This sample was inspired by a racf-l posting. Note that if the group is a universal group, only the users connected with some sort of extra privilege will be processed.

https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/IRRXUTIL/XGRPALU.txt

~~5.2~~5.6 XLGRES – Resume all users in a group

This sample was inspired by a racf-l posting. Note that if the group is a universal group, only the users connected with some sort of extra privilege will be processed.

https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/IRRXUTIL/XLGRES.txt

~~5.3~~5.7 XLISTGRP – Display group members alphabetically

This sample was inspired by a customer requirement. It takes a group name and displays the list of connected users in alphabetic order, with each user's name and group authority. Note that if the group is a universal group, only the users connected with some sort of extra privilege will be processed. ISPF tables are used for sorting, and so this sample must be run from within an ISPF environment.

https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/IRRXUTIL/XLISTGRP.txt

~~5.4~~5.8 XLISTUSR – Display a user's connected groups alphabetically

This sample was inspired by a customer requirement. ISPF tables are used for sorting, and so this sample must be run from within an ISPF environment.

https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/IRRXUTIL/XLISTUSR.txt

~~5.5~~5.9 XRACSEQ – Re-Implementation of RACSEQ in REXX

This sample re-implements the RACSEQ utility in REXX. RACSEQ is another RACF web page sample download. It is a utility, written in Assembly Language which calls the R_admin service to extract user and group profiles and display their contents to the screen. The intent was that the output could be captured by REXX programs using OUTTRAP() and be easier to parse than the output of the LISTUSER and LISTGRP commands. Of course, this means that a REXX re-implementation of RACSEQ is somewhat pointless as use of IRRXUTIL already makes profile data accessible to REXX programs. However, the intent of this sample is to demonstrate how a program can use the output of IRRXUTIL to figure out and organize the segment and field data of any profile returned by IRRXUTIL, without having any previous notion of what data will be returned.

https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/IRRXUTIL/XRACSEQ.txt

5.10 XRDEFROM – Copy a profile, including its member list

Inspired by a racf-l post, this exec performs an RDEFINE FROM that includes the member list. It could be modified to include any attribute(s) that RDEFINE FROM does not support.

https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/IRRXUTIL/XRDEFROM.txt

~~5.6~~5.11 XRLIST – Display sorted standard access list of a general resource profile

This sample was inspired by a customer requirement. The standard access list is sorted into user and group entries. The user entries are displayed first, in alphabetic order, with their names. Then the group entries are displayed in alphabetic order. ISPF tables are used for sorting, and so this sample must be run from within an ISPF environment.

https://github.https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/IRRXUTIL/XRLIST.txt

5.12 XRRSFPFX – Display RACF subsystem command prefix

Has anyone ever told you to issue a RACF operator command from the console, but you don't know the command prefix? Then you have to chase down your system programmer, or your SYS1.PROCLIB to figure it out? That's annoying!

Instead, just run this REXX exec.

https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/IRRXUTIL/XRRSFPFX.txt

5.13 XSETRAUD – Display only the auditing-related SETROPTS options

This was inspired by a customer requirement for a more granular SETROPTS LIST. Only the auditing SETROPTS options are displayed.

https://github.https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/IRRXUTIL/XSETRPWD.txt

~~5.7~~5.14 XSETRPWD – Display only the password-related SETROPTS options

This was inspired by a customer requirement for a more granular SETROPTS LIST. Only the password-related SETROPTS options are displayed. This output is augmented by an indication of whether password or password phrase enveloping is active. You can modify this program to display any old subset of SETROPTS LIST output: auditing options, Multi-Level Security options, data set options, etc.

https://github.https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/IRRXUTIL/XSETRPWD.txt

5.15 XSTARTED – "Health Check" the STARTED class

This sample examines all profiles in the STARTED class to make sure they are defined correctly with no references to undefined users or groups. See the file for all the details.

https://github.https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/IRRXUTIL/XSTARTED.txt

~~5.8~~5.16 XWHOCAN – Display the users who can modify a given profile

This sample takes a class and a profile name and indicates certain users who can modify the profile. Its main intent is to demonstrate the RACF group tree, but it checks for additional non-system-wide attributes as well. It is by no means exhaustive, nor is the output particularly pretty, but we wouldn't want to put Tivoli® zSecure out of business now, would we?

https://github.https://github.com/IBM/IBM-Z-zOS/tree/main/zOS-RACF/Downloads/IRRXUTIL/XWHOCAN.txt

6 References

For documentation on the IRRXUTIL interface, see z/OS Security Server RACF Macros and Interfaces.

For documentation on the R_admin (IRRSEQ00) callable service, see z/OS Security Server RACF Callable Services.

7 Disclaimers, etc.

This program contains code made available by IBM® Corporation on an AS IS basis. Any one receiving this program is considered to be licensed under IBM copyrights to use the IBM-provided source code in any way he or she deems fit, including copying it, compiling it, modifying it, and redistributing it, with or without modifications, except that it may be neither sold nor incorporated within a product that is sold. No license under any IBM patents or patent applications is to be implied from this copyright license.

The software is provided "as-is", and IBM disclaims all warranties, express or implied, including but not limited to implied warranties of merchantability or fitness for a particular purpose. IBM shall not be liable for any direct, indirect, incidental, special or consequential damages arising out of this agreement or the use or operation of the software.

A user of this program should understand that IBM cannot provide technical support for the program and will not be responsible for any consequences of use of the program.