

## Analyze Object Authorities Using Security APIs

[iPro Developer](#)

[Carsten Flensburg](#)

Carsten Flensburg

Tue, 07/31/2012 (All day)

APIs by Example

Do you sometimes wonder why a particular user can delete a certain object, or why a user can't access an object that every body else can? If you do, a security-related API named Retrieve User Authority to Object (QSYRUSRA) is designed to help answer these types of questions. In addition to retrieving a user profile's authority to a specific object, the QSYRUSRA API also returns information about the *source* of this authority. QSYRUSRA supports QSYS.LIB objects as well as objects in all other file systems.

The possible sources of authority include private authority, group authority, public authority, object authorization list, and any of the aforementioned authority sources obtained through adopted authority. The API, however, takes only adopted authority into consideration if authority is retrieved for the user profile special value \*CURRENT. Further, when applicable, the QSYRUSRA API provides authority information for all of a user profile's group profiles. In other words, if you ever have a question about users' authority to a specific object, QSYRUSRA can likely give you the answer.

To take advantage of the comprehensive and detailed object authority information that the QSYRUSRA API provides, I've written a CL command called Analyze Object Authorities (ANZOBJAUT). Additionally, I've made all this information readily available in both display and print format. The ANZOBJAUT command employs the QSYRUSRA API to access QSYS.LIB object authority information for a selected range of user profiles. The command also provides a couple of parameters that let you further narrow the presented list based on the actual source of authority and the level of authority granted (more about the ANZOBJAUT command shortly).

### The QSYRUSRA API Parameter Group

First, let me present the QSYRUSRA API parameter list, as defined in the IBM Information Center's API section (Figure 1).

**Figure 1: The QSYRUSRA API parameter list**

```
Retrieve User Authority to Object (QSYRUSRA) API

Required Parameter Group:

1 Receiver variable          Output   Char(*)
2 Receiver variable length   Input    Binary(4)
3 Format name                 Input    Char(8)
4 User profile name          Input    Char(10)
5 Qualified object name      Input    Char(20)
6 Object type                Input    Char(10)
7 Error code                 I/O     Char(*)

Optional Parameter Group 1:

8 ASP device                 Input    Char(10)

Optional Parameter Group 2:

9 Path name                  Input    Char(*)
10 Length of Path name       Input    Binary(4)

Default Public Authority: *USE
```

QSYRUSRA returns the object authority information to the *Receiver variable*, in the format specified by the

*Format name* parameter. Currently, only format USRA0100 is supported, but the authority information included in this format is exhaustive in most aspects, so no issues here. The second parameter specifies the length of the receiver variable, indicating to the API how much space is available. The *User profile name* whose authority should be returned is the fourth parameter; the special values \*PUBLIC and \*CURRENT are supported here. As mentioned earlier, the QSYRUSRA API will return information only about adopted authority if the user profile name \*CURRENT is passed.

When I specify a range of user profiles for the ANZOBJAUT command, I use the *Open List of Authorized Users (QGYOLAUS) API* to return the corresponding list of user profiles, and then call the QSYRUSRA API for each user profile as I process the list. The fifth and sixth parameters, *Qualified object name* and *Object type*, respectively, identify the object for which to retrieve authority information. The final required parameter is the standard API error code.

Next follows two optional parameter groups. The first group is a single parameter defining the *ASP device* in which the specified object resides. I chose not to expose this parameter in the ANZOBJAUT command interface, but given the requirement, you could easily do so.

The second optional parameter group lets you target the QSYRUSRA API against an object in the IFS that's not located in QSYS.LIB. If you indicate the *Path name* and *Length of path name* parameters, you must also specify a qualified object name of \*OBJPATH and blanks for the object type. The IFS object path name must be passed as a Qlgl\_Path\_Name\_T structure that contains a path name or a pointer to a path name. For more information about this structure, see the recent APIs by Example article "Zip and Unzip Files with the New 7.1 Zip API Support," listed in Find Out More, below.

The ANZOBJAUT Command

You can see what the ANZOBJAUT command prompt looks like in Figure 2.

Figure 2: Analyze Object Authorities (ANZOBJAUT) command prompt

```
-----
                                Analyze Object Authorities (ANZOBJAUT)

Type choices, press Enter.

Object . . . . .                               Name
Library . . . . .                               *LIBL      Name, *LIBL, *CURLIB
Object type . . . . .                           *ALRTBL, *AUTL, *BNDDIR...
User profile . . . . .                           *ALL          Name, generic*, *ALL...
Select . . . . .                               *ALL          *ALL, *AUT, *AUTL, *GRP, *PVT
Authority . . . . .                             *ANY          *ANY, *ALL, *CHANGE, *USE...
Output . . . . .                               *              *, *PRINT

-----
```

Although I explain the command and all its parameters in the associated help text panel group, I'll briefly walk you through the essential parameters here. (For instructions on how to create the command, see "How to Compile," below.) Obviously, the object and object type identify the object for which the authority analysis is performed. You can then specify a specific user profile, a generic user profile, or the special value \*ALL or \*CURRENT to identify which user profiles to investigate in the analysis.

The SELECT parameter lets you narrow the resulting list into one of five categories:

- List all user profiles.
- List only user profiles having some sort of access to the object.
- List only user profiles whose authority comes from an authorization list.
- List only user profiles whose authority comes from one or more group profiles.
- List only user profiles being granted a private authority.

Likewise, using the OBJAUT parameter, you can specify that only user profiles having a specific authority level are included in the list:

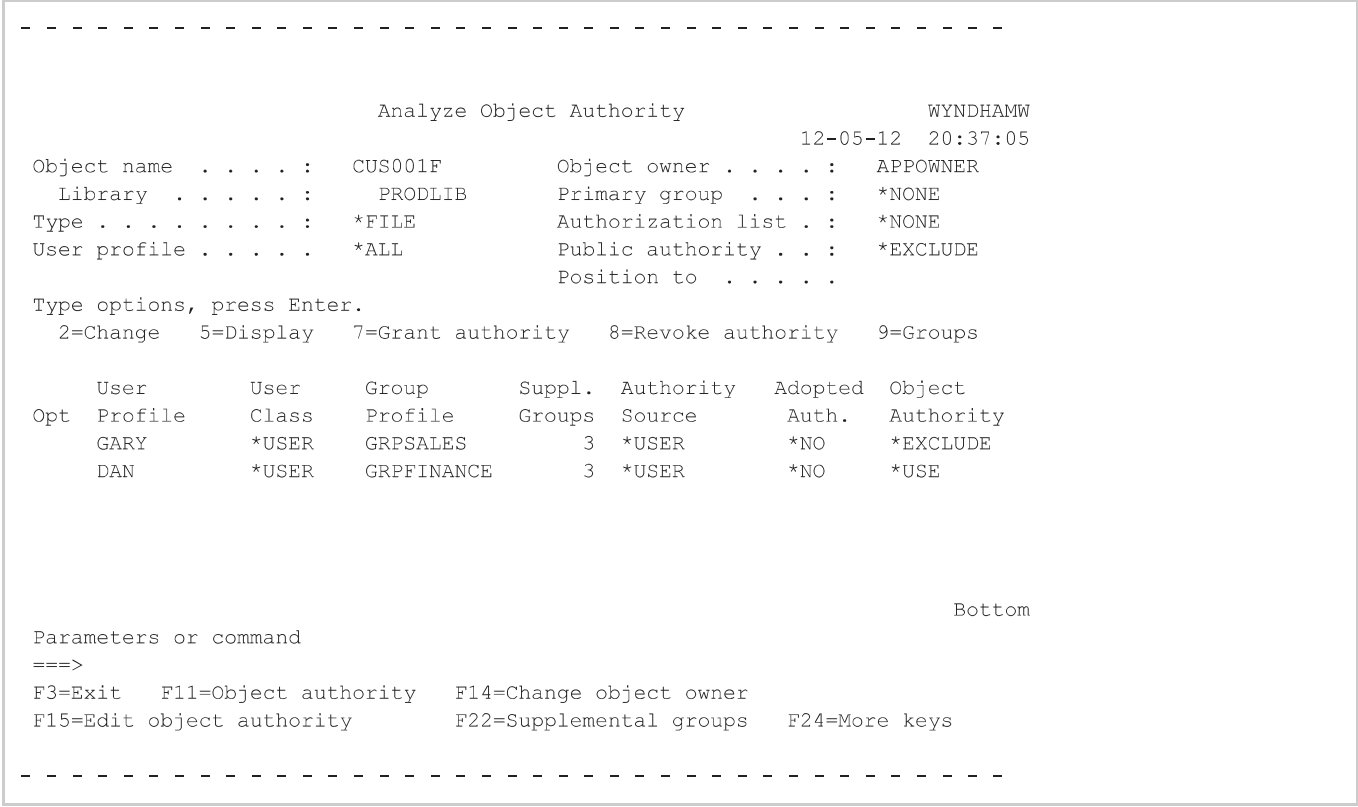
- Include all user profiles, irrespective of their authority.
- Include only user profiles having \*ALL authority.
- Include only user profiles having a minimum of \*CHANGE authority.
- Include only user profiles having a minimum of \*USE authority.
- Include only user profiles having \*EXCLUDE authority.

The final command parameter defines whether the list of user profiles should appear in a list panel or print with your job’s spooled output. To give you an example of how the ANZOBJAUT command’s list panel looks, I ran the following command:

```
ANZOBJAUT OBJ (PRODLIB/CUS001F)
          OBJTYPE (*FILE)
          USRPRF (*ALL)
          SELECT (*PVT)
          OBJAUT (*ANY)
```

The above command will list all user profiles having private authority of any kind to the file CUS001F in library PRODLIB. Figure 3 shows the resulting list panel, which also includes cursor-sensitive help text to explain the different parts of the panel as well as the list columns, list options, and function keys.

Figure 3: Analyze Object Authority list panel



For now, I’ll just point out that list option 9=Groups opens a window, listing all group profiles contributing to the accumulated group authority granted to the user profile, as well as the actual object authority and the source of this authority. Using the cursor position and the F22 key in the group list will call the ANZOBJAUT command for the group profile selected; doing so also provides detailed information about the object authority available to the group profile.

How to Compile

Below, you’ll find instructions on how to create the Analyze Object Authorities command. The following sources are included with the code download associated with this article:

- CBX253—RPGLE: Analyze Object Authorities—CPP
- CBX253C—RPGLE: Analyze Object Authorities—Choice Program

CBX253E—RPGLE: Analyze Object Authorities—UIM Exit Program

CBX253U—RPGLE: Analyze Object Authorities—UIM Cond Program

CBX253V—RPGLE: Analyze Object Authorities—VCP

CBX253S—RPGLE: Analyze Object Authorities—Services

CBX253B—SRVSRG: Analyze Object Authorities—Binder source

CBX253L—RPGLE: Retrieve Command Parameter Value List

CBX253H—PNLGRP: Analyze Object Authorities—Help

CBX253P—PNLGRP: Analyze Object Authorities—Panel group

CBX253X—CMD: Analyze Object Authorities

CBX253M—CLP: Analyze Object Authorities—Build command

To create all above command objects, compile and run the CBX249M CL program, following the instructions in the source header. You'll also find compilation instructions in the respective source headers of the individual sources.

## Find Out More

- [CMD—Possible choices and values for parameter definitions](#)
- [UIM—Exit Program for Conditioning Panel Items](#)

## IBM i 7.1 Information Center documentation

- [Open List of Authorized Users \(OGYOLAUS\) API](#)
- [Retrieve User Authority to Object \(OSYRUSRA\) API](#)
- [Retrieve User Information \(OSYRUSRI\) API](#)
- [Resource security—IBM i](#)
- [Security Guide for IBM i 6.1 Redbook](#)
- [Security-related APIs](#)

## Articles at *iPro Developer*

- ["APIs by Example: Zip and Unzip Files with the New 7.1 Zip API Support"](#)
- ["Carsten's Corner: Compare Object Authority for a Group of Objects"](#)
- ["Carsten's Corner: New Utility to Compare Object Authority"](#)
- ["Writing RPG Code Then and Now"](#)

**Source URL:** <http://iprodeveloper.com/application-development/analyze-object-authorities-using-security-apis>