

[print](#) | [close](#)

APIs by Example: Security APIs - and Transfer of User Object Ownership

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 04/22/2010 (All day)

The Security API topic is one of the most comprehensive in the IBM i Information Center API section. In release 6.1, the security APIs are divided into eight subcategories, including Security-related APIs, Digital Certificate Management APIs, User Function Registration APIs, and Validation List APIs, to mention a few. And each subcategory comprises many APIs. One of the largest groups is the Security-related APIs, from which I've picked an API for today's issue of APIs by Example.

The API in question was given the rather lengthy name of List Objects a User Is Authorized to, Owns, or Is Primary Group of (QSYLOBJA) API. The API capacity demonstrated here relates to object ownership. For a specified user profile the QSYLOBJA API lists all objects owned by the user profile to a user space. From there on, the object list is processed using the Retrieve Pointer to User Space (QUSRPTRUS) API and pointer arithmetic. In this API example I use the QSYLOBJA API-produced object list as the foundation for a CL command that lets you transfer object ownership from one user profile to another.

The QSYLOBJA API is capable of listing objects in the QSYS.LIB file system and the IFS (i.e., objects in a library and objects in a directory, respectively). Due to the difference in naming and qualification of these two types of objects, you will, however, need to do either one or the other when you call the API. The object type is implied in the return format name, as documented in the following list of return formats:

- OBJA0100: Each entry contains the object name, library, type, authority holder indicator, ownership indicator, auxiliary storage pool (ASP) device name of library, and ASP device name of object.
- OBJA0110: This format only returns path names for objects in a directory. Each entry contains the offset to the path name, the length of the path name, type, authority holder indicator, ownership indicator, ASP device name of object, and the path name value.
- OBJA0200: Each entry contains the same information as format OBJA0100 plus the authority values.
- OBJA0210: This format only returns path names for objects in a directory. Each entry contains the same information as format OBJA0110 plus the authority values.
- OBJA0300: Each entry contains the same information as format OBJA0200 plus the object attribute and descriptive text.
- OBJA0310: This format only returns path names for objects in a directory. Each entry contains the same information as format OBJA0210 plus the attribute and descriptive text.

As for the QSYLOBJA API's remaining parameters, here's an excerpt from the release 6.1 API documentation in the Information Center documenting the API interface in its entirety:

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	User profile name	Input	Char(10)
4	Object type	Input	Char(10)
5	Returned objects	Input	Char(10)
6	Continuation handle	Input	Char(20)
7	Error code	I/O	Char(*)

Optional Parameter Group:

8	Request list	Input	Char(*)
---	--------------	-------	---------

The *Qualified user space name* designates the user space to which the API output is directed. The *User profile name* parameter defines the user profile for which the object list is generated. In the context of this API, this implies that the user profile is either the object's owner, primary group profile, or is privately authorized to the object. Which qualification applies is defined by the fifth parameter, *Returned objects*, in conjunction with the final and optional parameter, *Request list*. The *Returned objects* parameter allows the following special values to be specified:

*OBJAUT	The list of objects the user is authorized to is returned.
*OBJOWN	The list of objects the user owns is returned.
*BOTH	The list of objects the user is authorized to and owns is returned.
	The list of owned objects precedes the list of authorized objects.
*REQLIST	The values specified in the request list parameter is used.

At the time the QSYLOBJA API was introduced, only object ownership and object private authorization were applicable because, at that point, the concept of object primary group had not yet been implemented. When the object primary group attribute was introduced, the optional Request list parameter was added as well as the Returned object parameter special value *REQLIST. The latter indicating that the former would hold the actual values qualifying which object access relations to include in the API list output.

Consequently the Request list parameter allows for an array of 10-character values to be specified, thus providing for any combination of the following special values, including the *OBJPGP special value added to support object primary group qualification:

*OBJAUT	Returns the list of objects the user is authorized to.
*OBJOWN	Returns the list of objects the user owns.
*OBJPGP	Returns the list of objects that the user is the primary group for.

The object type parameter enables you to limit the list of objects to the specified object type only. For directory objects, file system object types such as *DIR and *STMF are supported, defining directories and stream files, respectively. Beyond the *ALLIFS special value targeting all directory

objects, the TFROBJOWN command supports only the *DIR and *STMF special values but is easily enhanced in case other IFS object types would require individual processing.

The *Continuation handle* parameter originates in the circumstance that the maximum size of a user space is 16,776,704 bytes, or approximately 16MB. In the event that the returned list exceeds this limit, the QSYLOBJA API returns a continuation handle in the Header Section of the data written to the user space. Specifying this continuation handle on a subsequent call to the API will cause it to continue the object list where it left off on the previous call, as opposed to a blank continuation handle leading to the list being built from the top. In essence, what you need to do is keep calling the QSYLOBJA API and process the object list returned as long as a valid continuation handle is returned and until the object list is exhausted.

The presence of a valid continuation handle is signaled by the letter "P" in the *Information Status* field in the user space generic header section. The letter "P" in this context translates to: "The information returned in the user space is valid but incomplete," (i.e., **Partial**). An information status of "C" indicates that the returned list is valid and complete, while a status of "I" is returned in case the data in the list is invalid and the continuation handle therefore is undefined.

The header section containing the continuation handle also includes the user profile name actually used for building the list. If the special value *CURRENT was specified for the User profile name input parameter, this field will contain the name of the user profile resolved. Additionally a *Reason code* is available in the header section, specifying if the choice of return format caused any objects qualifying for the list to be excluded. This would be the case if for example directory objects were found but the return format specified support library objects only.

The *API error code parameter* has been discussed and explained in great detail in articles previously published, so I've included links to a couple of these at the end of this article. As for the exercise of processing List API output in a user space, a walkthrough of this procedure is offered in the APIs by Example article [Retrieve Subsystem Entries API](#) to which a link is included below also. To see for yourself how the pieces fit together, if in doubt, I suggest you run the TFROBJOWN command processing program CBX214 in the source debugger, while the command is executing.

Speaking of which, the TFROBJOWN command prompt panel has the following appearance, including a conditional parameter at the end:

Transfer Object Owner (TFROBJOWN)		
Type choices, press Enter.		
Object type	*ALL, *ALLLIB,	
*ALLIFS...		
Current owner	Name	
New owner	Name	
Current owner authority	*REVOKE	*REVOKE, *SAME
Omit object type	*NONE	*NONE, *ALRTBL,

```
*AUTL...
+ for more values
```

The *Omit object type* parameter applies to library objects and is therefore only displayed if the specified Object type warrants objects of this type to be selected. The Object type parameter, in addition to regular object types such as *FILE, *PGM, and *DTAQ, supports the following special values:

*ALL	All objects, both library and directory objects, are processed
*ALLLIB	All library objects (QSYS.LIB) are processed
*ALLIFS	All directory (IFS) objects are processed

You specify the name of the user profile whose object ownership should be transferred for the Current owner parameter and the user profile receiving object ownership for the New owner parameter. The Current owner authority parameter defines whether the current owner should retain the current private authority to the object, following the ownership transfer. For library objects, you further have the option of specifying up to 10 object types to omit when performing the object ownership transfer. The command and all its parameters are also documented in detail in the accompanying help text panel group.

Here's an example of how it would look if you wanted to transfer all library objects except message queues and user profiles owned by user profile USERA to user profile USERB:

```
TFROBJOWN OBJTYPE( *ALLLIB )
           CUROWN( USERA )
           NEWOWN( USERB )
           CUROWNAUT( *REVOKE )
           OMITTYPE( *MSGQ *USRPRF )
```

Following the successful execution of the above command, you'll receive a completion message indicating how many of the selected objects whose ownership were transferred correctly and how many, if any, objects that failed in the attempt to transfer their ownership.

The TFROBJOWN command, depending on the object type of the object in question, employs the Change Object Owner (CHGOBJOWN) and the Change Owner (CHGOWN) commands to perform the ownership transfer and leaves the completion or diagnostic messages issued by these commands in the job log of the job running the TFROBJOWN command. Using the Display Job Log (DSPJOBLOG) command, you then have the option of investigating the exact outcome following the execution of the TFROBJOWN command.

In the event one of the change commands is failing, the diagnostic and exception messages generated are also returned as diagnostic messages to the caller of the TFROBJOWN command, preceding the aforementioned completion message.

This APIs by Example includes the following sources:

```
CBX214    -- RPGLE    -- Transfer Object Owner - CPP
CBX214H   -- PNLGRP   -- Transfer Object Owner - Help
CBX214V   -- RPGLE    -- Transfer Object Owner - VCP
CBX214X   -- CMD      -- Transfer Object Owner

CBX214M   -- CLP      -- Transfer Object Owner - Build command
```

To create all the TFROBJOWN command objects, compile and run the CBX214M program, following the instructions in the source header. You can also find compilation instructions in the respective source headers.

This APIs by Example article is based on a suggestion submitted by Peter Kemp, of Australia. If you have any ideas or suggestions for me to cover in future APIs by Example articles, please forward these to me at flensburg@novasol.dk.

Related Articles:

[APIs by Example: Retrieve Subsystem Entries API](#)

[APIs by Example: Check Object Authority](#)

[Getting Started with APIs, Part 2—Error Handling](#)

[APIs by Example: Using the ERRCo200 Data Structure](#)

This article demonstrates the following Security APIs:

[List Objects a User Is Authorized to, Owns, or Is Primary Group of \(QSYLOBJA\) API](#)

[Security-related APIs](#)

[Security APIs](#)

[Retrieve the source code for this API example.](#)

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-security-apis-and-transfer-user-object-ownership>