

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

APIs by Example: Authorization List APIs and Secured Objects

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 07/27/2006 (All day)

Authorization lists offer an effective and flexible method of securing sensitive objects and data. With a single authorization list, you can manage access to many objects and, for data files, this management can take place even while the file is open and in use. For these and other reasons, authorization lists are instrumental in most contemporary security models. Today I demonstrate a couple of APIs that retrieve information about authorization lists.

One of these APIs was recently added to the security-related API category. As a matter of fact, V5R4 is the first release to include the Retrieve Authorization List Information (QSYRTVAI) API in the base install. But even before the general availability of V5R4, this API was available through PTFs for V5R2 and V5R3. It's also included in recent CUM packages for both releases. Please follow the links at the end of this article for more information about these PTFs.

The purpose of the QSYRTVAI API is to enable monitoring of the number of objects currently secured by an authorization list, as well as the remaining capacity of the authorization list. An authorization list is currently capable of storing 2,097,104 object entries per auxiliary storage pool (ASP). Before the QSYRTVAI API's advent, the only way to find out that this limit was about to be exceeded was when the MCH2804 exception message occurred when one tried to add a new object. Even though 2,097,104 objects sounds like a lot, it's important to take into account that for multimember files, each member counts as one object in the authorization list.

Anyway, to show an example of the QSYRTVAI API in action, I wrote the Check Authorization List Size (CHKAUTLSIZ) command. Here's the CHKAUTLSIZ command prompt:

Check Authorization List Size (CHKAUTLSIZ)

Type choices, press Enter.

Authorization list

Name

If you run the following command from a command line

```
CHKAUTLSIZ AUTL(TESTAUTL)
```

it returns a completion message similar to this one:

```
For TESTAUTL ASP *SYSBAS 41232 entries in use and 2055872 available.
```

I have included the important paragraphs of the API documentation in the CHKAUTLSIZ command's help panel group, to make this crucial information readily available.

Now that I can get the authorization list object count, retrieving the actual individual object names might also be interesting. Although the Display Authorization List Objects (DSPAUTLOBJ) command is already available for this purpose, it requires you to use an output file to get at the information programmatically and does not include information about objects in directories in the IFS. On my V5R3 system, the CPI221C informational message appears (following the IBM copyright notice; you have to scroll down or press F11 to actually see it) if such objects are secured by the authorization list for which the DSPAUTLOBJ command is run. For example, if you scroll down, you might see the following message:

```
6 objects were not included in this list.
```

If you place the cursor on the message line and press F1, the second-level help text is displayed. Here's what that looks like:

```
Message . . . . . 6 objects were not included in this list.
Cause . . . . . Objects were found that meet the search criteria
but were not included in the returned list. Objects in a directory
could not be processed.
Recovery . . . . . Use the QSYLATLO API with format ATLO0110,
ATLO0210, ATLO0300, or ATLO0400 to retrieve objects in a directory.
```

If you want to know exactly which directory objects are secured by the authorization list, IBM suggests that you use the List Objects Secured by Authorization List (QSYLATLO) API. So that is what I do. The API documentation reveals that only four parameters are required and that the API returns the requested information to a user space. And further, the documentation states that the QSYLATLO API has a number of different return formats, depending on the extent of the information to return, as well as the file systems included. Some formats include either the QSYS.LIB, the QDLS or the remaining IFS file systems, or a combination of two or all file systems. Please check the aforementioned API documentation for all the details (a link to the API documentation is at the end of this article).

Because the QSYS.LIB and QDLS file system objects are returned in a different format than the formats of the other (IFS) file systems, I've included two sample programs, one for each variant. The QSYS.LIB and QDLS return format is simple to process, because it follows the common standard of adding a fixed entry length to jump from one list entry to the next. But the directory objects format is a bit more complex.

Whereas the QSYS.LIB and QDLS objects are identified by a 10-byte object name, a 10-byte library name, and a 10-byte object type, the IFS objects are identified by a name of varying length. The return format for these objects is therefore made up of a fixed-length data structure, which provides the offset to a path name data structure, in turn including a path name of varying length. Here's the ATLO0210 entry data structure:

```
**-- Authorization list IFS object entry
D ATLO0210          Ds                      Qualified
D                                     Based( pLstEnt )
D  OfSPthNam                10i 0
D  LenPthNam                10i 0
D  ObjTyp                  10a
```

D	AutHlrr	1a
D	ObjOwn	10a
D	ObjAttr	10a
D	TxtDsc	50a
D	ObjPgp	10a
D		1a
D	AspDev	10a

The space pointer, which defines the location of the ATLOO200 data structure, is initially set to the user space list offset address provided by the user space generic header information:

```
pLstEnt = pUsrSpc + UsrSpcHdr.OfsLst;
```

And here's the path name data structure, which is located at the offset specified by the OfsPthNam subfield in the preceding example and has the length specified by the adjacent LenPthNam subfield:

```

**-- API path
D Qlg_Path_Name    Ds                      Qualified
D                                     Based( pQlg_Path_Name )
D CcsId            10i 0
D CtrId            2a
D LngId            3a
D                  3a
D PthTypI          10i 0
D PthNamLen        10i 0
D PthNamDlm        2a
D                  10a
D PthNam           5000a

```

The Qlg_Path_Name structure is also based on a pointer, which is set by the following statement. Notice that the Qlg_Path_Name offset that the QSYLATLO API returns is calculated from the beginning of the user space, as opposed to the beginning of the ATLOO210 list entry.

```
pQlg_Path_Name = pUsrSpc + ATLOO210.OfsPthNam;
```

The *PthNam* varying-length variable that stores the retrieved path name is then updated by the following statement:

```

PthNam = %Subst( Qlg_Path_Name.PthNam
                1
                Qlg_Path_Name.PthNamLen
                );

```

Variable *PthNam* now holds the actual IFS object name (and length), and after this information has been passed back to the caller in a message, the list processing continues. To advance to the next list entry, I need to calculate the distance in bytes between the current list entry and the next, which the following statement takes care of:

```

pLstEnt += UsrSpcHdr.SizLstEnt +
           ( %Size( Qlg_Path_Name ) -
             %Size( Qlg_Path_Name.PthNam ) ) +
           Qlg_Path_Name.PthNamLen;

```

As I mentioned earlier, this calculation deviates from the usual simple addition of a fixed entry length because of the variable length of the path name, so let me explain what I'm doing.

1. Before the preceding calculation, the pLstEnt pointer that the ATLOO210 data structure is based on points to the beginning of the current list entry.
2. The length of the fixed part of the ATLOO210 data structure is defined by the List Entry Size specified in the user space generic header information. This length is added to pLstEnt:

```
pLstEnt += UsrSpcHdr.SizLstEnt
```

3. The next length that I need to add is the fixed part of the path name structure. This length is calculated by subtracting the length of the PthNam subfield from the total length of the Qlg_Path_Name data structure:

```
+ ( %Size( Qlg_Path_Name ) - %Size( Qlg_Path_Name.PthNam ) )
```

4. And finally, I add the actual length of the current path name to arrive at the beginning of the next entry:

```
+ Qlg_Path_Name.PthNamLen
```

To avoid referring to an address outside of the user space list, the calculation is performed only as long as there are more list entries to retrieve. The actual number of list entries is also specified in the user space generic header.

To call the sample programs, specify the following statements from a command line (or use the SBMJOB command to submit the execution to batch):

```
Call      Pgm( CBX1592 )  Parm( ' ' )
```

- returns IFS directory objects for the specified authorization list.

```
Call      Pgm( CBX1593 )  Parm( ' ' )
```

- returns QSYS.LIB and QDLS objects for the specified authorization list.

Both sample programs return a message to the caller for each object retrieved, as well as a final completion message specifying the total number of objects retrieved. For batch jobs, the messages appear in the job log.

Now that I can retrieve IFS directory objects secured by a specific authorization list, I have also created the building blocks required to create my own Display Authorization List IFS Objects (DSPAUTLIFS) command that is currently missing. That task will be the focus of the next APIs by Example.

This APIs by Example includes the following sources

```
CBX1591  -- Check Authorization List Size - CPP
CBX1591H -- Check Authorization List Size - Help
CBX1591X -- Check Authorization List Size
```

```
CBX1592  -- Retrieve Authorization List IFS Objects - example
CBX1593  -- Retrieve Authorization List QSYS Objects - example
```

Compilation instructions are in the source headers, as usual.

For the CHKAUTLSIZ command to run on releases V5R2 and V5R3, the following PTFs need to be installed. For both releases, the PTFs in question are included in CUM packages from early this year, C6045520 and C6080530, respectively.

- [The QSYRTVAI API V5R2 & V5R3 APAR](#)
- [The QSYRTVAI API V5R2 & V5R3 QSYSINC include APAR](#)
- [The QSYRTVAI API V5R3 PTF cover letter](#)

This article demonstrates the following APIs

Retrieve Authorization List Information (QSYRTVAI) API

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/apis/qsyrtsvai.htm>

List Objects Secured by Authorization List (QSYLATLO) API

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/qsylatlo.htm>

Send Program Message (QMHSNDPM) API

<http://publib.boulder.ibm.com/infocenter/iserics/v5r3/topic/apis/QMHSNDPM.htm>

You can retrieve the source code for this API example from the following link:

http://www.pentontech.com/IBMContent/Documents/article/52913_91_AuthList.zip

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-authorization-list-apis-and-secured-objects>