

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

Analyze Your Audit Journal with RPG

[System iNEWS Magazine](#)

[Carsten Flensburg](#)

Carsten Flensburg

Mon, 02/01/2010 (All day)

[Click here](#) to download the code bundle.

To report code errors, email [SystemiNetwork.com](mailto:info@SystemiNetwork.com)

Depending on how your system audit journal QAUDJRN is configured, it contains a wealth of information that not only reveals ongoing

system activity, but also documents possible attempts to compromise system security and access controls. Given this comprehensive source of audit and security information and the enormous amount of data often available, the challenge quickly becomes the task of filtering out the events and actions of specific interest to security officers responsible for system security in general, and of course to internal and external system auditors in particular. This article explains how to filter out those events and actions using the preferred Analyze Audit Journal (ANZAUDJRN) command.

Audit Tools on Board

IBM i has available to it a number of native audit tools in the form of CL commands, including the Display Journal (DSPJRN) and Display Audit Journal Entry (DSPAUDJRNE) commands, which have been in service for quite a while, as well as the latest addition to the collection: the Copy Audit Journal Entry (CPYAUDJRNE) command. The latter especially is useful when you're performing an investigation because it maps all the information in the journal entry's entry-specific data section into individual data fields. This, in turn, lets you interrogate the entry event type, name, library, and type of object associated with the journal entry and other crucial entry-specific information.

The CPYAUDJRNE does, however, as its name implies, copy the retrieved information to an output file that you must process using Query/400 or a similar tool—turning the investigation into a two-step procedure. Too, the CPYAUDJRNE command lacks some of the parameters available on the DSPJRN command, eliminating the option of excluding journal entries up front based on, for example, job name or program name. What fascinates me about encountering situations like this is that the comprehensive toolset available on IBM i lets me easily devise and build utilities that meet my specific needs—doing so even on a tight budget. All it takes is the ILE RPG compiler and the API manual.

When it comes to processing journal entries, though, options other than APIs exist. For realtime processing of journal entries I've often used the Receive Journal Entry (RCVJRNE) command, which makes it possible to retrieve and process journal entries as they arrive in the journal receiver. There's also a Retrieve Journal Entry (RTVJRNE) command targeted at extracting journal entries into a CL program. However, for the purpose of specifying an arbitrary subset of journal entries, the Retrieve Journal Entries (QjoRetrieveJournalEntries) API is hard to beat, both in terms of the extensive parameter set offered as well as the speed by which it processes and returns the journal entries selected.

API Documentation—First & Second Glance

At first glance, the QjoRetrieveJournalEntries API parameters ([Figure 1](#)) seem to be quite surmountable. It's the second glance that makes a slightly intimidating impression when you try to make heads or tails of the *Journal entries to retrieve* entry selection variable length parameter with all its optional substructures. However, using this parameter you are able to specify the same set of selection criteria as those offered by the DSPJRN and RCVJRNE commands. In IBM i 5.4 this amounts to 22, and in 6.1 it is 23 optional entry selection criteria.

As you will see once you get the documentation deciphered, it's pretty straight forward to code both the API prototype and the entry selection parameter data structures due to the consistency of the variable length parameter structures. [Figure 2](#) displays the QjoRetrieveJournalEntries API prototype and [Figure 3](#) the data structures defining the entry selection parameter set. Coding the entry selection data structures this way makes it easy to snap in and snap out each individual entry selection criteria. [Figure 4](#) shows how the API call is performed and (hopefully) demonstrates the aforementioned statement.

Also note the API documentation's use of the term *omissible* regarding the fifth and sixth API parameter. To get the prototype compliant with the API behavior you must use the *Options* keyword with the special value *Omit for these parameters—as opposed to the more commonly employed convention of *optional* parameter groups defined by Options(*NoPass). The latter implies that you can simply leave out the parameter on the API call, but if you do, no other parameter belonging to the same or following optional parameter groups can be specified. For omissible parameters you must specify the special value *Omit in place of the parameter, thus allowing subsequent parameters to be specified, irrespective of the one omitted.

The ANZAUDJRN Command—Putting the Pieces Together

Now that you have the core of the utility under construction in place and the API prototype and associated parameter structures defined, focus your attention on the command interface. Using the DSPJRN command parameters as a starting point, add selection criteria for entry event type, object name, type, and library as well as IFS path name. This covers most of the requirements originally envisioned, and the combined set of audit journal entry filter options should make it significantly easier to target the journal entries of particular interest in any given situation.

To also provide for a specific level of generic search, I've completed the command interface design with a final parameter that lets you specify a varying-length character string on which to perform a match against the audit journal entry's Entry Specific Data (ESD) part. You can either specify a specific location within the ESD in the form of an offset value or have the command scan the whole ESD for the search argument given. You can see the resulting Analyze Audit Journal (ANZAUDJRN) command interface in [Figure 5](#).

While most of the ANZAUDJRN command's parameters form a one-to-one relationship with the entry selection criteria offered by the QjoRetrieveJournalEntries API and therefore needs little reflection, there are other concerns, including the available and valid range of parameter input values. The command's vital *Entry type* parameter calls for a source that provides the list of entry type codes and their description for the current release of the operating system. Such a list would offer the foundation not only for checking the entry type parameter input, but also for allowing the parameter's choice text and F4 list function to be dynamically created for the command prompt. This would make it a bit easier for the people using the command to decide what their options are and the correct value to enter.

Other command parameters, such as the object and path related ones, need to be validated in terms of presence in the journal entry ESD to determine whether they should be included in the command prompt, depending on the actual *Entry type* entered. To sum up the requirements established so far, I need to determine the following:

- Which audit journal entry types are supported for the current release
- The textual description of each audit journal entry type
- The presence and location in the ESD of object type, name, and library
- The presence and location in the ESD of an IFS object path name

Overcoming the Obstacles

As for the first challenge, it's documented in the "System i Security Reference" manual's Appendix F that for each audit journal entry type there's a corresponding system model file following the naming convention QASYxxJ5 for output files of format *TYPE5, where xx is replaced by the entry type. Based on this information, it's possible to use the *Open List of Objects* (QGYOLOBJ) API to list all QASYxxJ5 files and retrieve each entry type supported.

The entry type textual description also requires a sort of "undocumented feature" approach. Coincidentally, a while back I came across the range of message descriptions in the system QCPFMSG message file employed to store both the journal codes and the journal entry types, as well as their respective textual descriptions. Exploiting this knowledge, I can use the *Retrieve Message Description* (QMHRTVM) API to retrieve the entry type description.

Based on an earlier observation, you also can solve the final problem of locating object information in the journal entry ESD. All audit journal entry type formats are laid out in the system model files, and IBM took the care to enforce a naming convention for the fields defined in the record formats. The fields containing object name, library, and type, as well as IFS object path name, path name CCSID, and path name length, are named using the same name pattern. The object name, library, and type resolve into xxONAM, xxOLIB, and xxOTYP, respectively, where xx is (again) replaced by the journal entry type supported by the file format in question. Listing the fields of a file's record format as well as the fields' location within the record format is something that the *Retrieve database file description* (QDBRTVFD) API is capable of.

Making the Ends Meet

It seems that a couple of the current model files have escaped IBM's design review board, but I've taken care of the few exceptions found. Once that journal entry data collection is completed, I have the information needed to control the ANZAUDJRN command prompt, parameter validation, and entry selection:

- Valid journal entry types are displayed as journal entry type parameter choice text and F4 prompt list, the latter including the entry type descriptive text, and all performed by choice program CBX607C. [Figure 6](#) displays how the F4 prompt list looks on an IBM i 5.4 system.
- The journal entry type list is employed in validity checking program CBX607V to perform the validation of the entry type specified.
- Prompt control program CBX607P ensures that the object name, library, and type parameter is only displayed on the command prompt if the chosen entry type contains object information.
- The IFS path name parameter is only displayed on the command prompt if the chosen entry type contains an IFS object path name; again managed by prompt control program CBX607P.

- The object name, library, and type as well as IFS object path name is located using the record format field offset values, and the corresponding journal entry values are used in the entry selection process by command processing program CBX6071.

I also should mention that to minimize the performance impact of the above information retrieval procedure, I load and store all data in a user index object and provide functions to retrieve the appropriate attributes wherever and whenever needed. Every time the ANZAUDJRN command is run, the presence of the user index and the release level associated with its content is checked. The user index and journal entry information is subsequently rebuilt if necessary, but only then.

While the above approach exposes you to the risk of IBM changing its implementation of the resources involved, it does provide a speedy and accurate outcome given the circumstances at hand. Another feasible path would be to manually acquire and load the information into the user index and then verify possible changes following each release upgrade. If the latter option makes a more appealing impression and you need assistance in making it work, feel free to contact me.

It Works—What to Do Next?

The ANZAUDJRN command's main command processing program (CPP) is quite simple. Once you've set up the parameters for the QjoRetrieveJournalEntries API call, it performs most of the hard work (taking into account the number of journal entries) at an impressive speed. For each journal entry retrieved, the selection criteria not directly supported by the API are evaluated by the CPP, and all entries accepted are added to the User Interface Manager (UIM) list panel for either display or print. The UIM is beyond the scope of this article, but I've discussed the mechanics of UIM in earlier articles and included links to these and other relevant articles below.

To create and run the ANZAUDJRN command, follow the steps documented in "Command ANZAUDJRN Source Specification" below; and once the task is completed, take it for a test ride. To find out how many times and by whom the command CHGUSRPRF had been run for the duration of the time covered by the entire journal receiver chain, I ran the following command:

```
ANZAUDJRN ENTTP(CD) RCVRNG(*CURCHAIN) OBJ(CHGUSRPRF) OBJLIB(QSYS)
OBJTYPE(*CMD)
```

Next, I ran the ANZAUDJRN command to find out how many changed passwords failed to pass the password composition (QPWD*) system values:

```
ANZAUDJRN ENTTP(CP) EVTTP(A) RCVRNG(*CURCHAIN) SEARCH(*SYSVAL 358)
```

[Figure 7](#) shows what the resulting list panel looks like. The list panel lets you execute the Display Journal (DSPJRN) command, positioning the entry list produced at the journal entry for which option 5 was specified. Likewise, you can run the Work with Job (WRKJOB) command for the job that deposited the selected journal entry, and finally the Work with User Profile (WRKUSRPRF) command for the user profile associated with the journal entry.

A number of function keys also are available to provide shortcuts to relevant commands and list actions, including the Display Security Auditing (DSPSECAUD) command, the Work with Journal Attributes (WRKJRNA) command displaying the attributes of the system audit journal (QAUDJRN), as well as a command line window. Function key F11 toggles between the three list views included. In addition to the above offering you also can view program name, program library, remote port, and remote IP address as well as object type and object name or object path.

End of Story

The ANZAUDJRN command is a handy tool when you need to track down specific types of user or program activity on your system or want to investigate critical security events. In Find Out More below, I've included links to articles explaining how to set up the system audit journal to provide for the handling of such situations. As Scouts say: "Be prepared." That's half the deal when it comes to auditing. Hopefully the ANZAUDJRN command will help you cover the other half.

Carsten Flensburg is a **System iNEWS** technical editor and has been a System i programmer since 1992. His focus areas are modular system design, API programming, system integration, and communication. Carsten currently works as a System i application development manager for a European vacation rental company called Novasol, which is a part of the U.S.-based Wyndham Worldwide Corporation.

Command ANZAUDJRN Source Specification

The following source members are involved in creating the ANZAUDJRN command:

Member	Type	Text
CBX607	RPGLE	Analyze Audit Journal-Services
CBX607B	SRVSRCL	Analyze Audit Journal-Binder Source
CBX607C	RPGLE	Analyze Audit Journal-Choice Program
CBX607P	RPGLE	Analyze Audit Journal-PCP
CBX607V	RPGLE	Analyze Audit Journal-VCP
CBX607H	PNLGRP	Analyze Audit Journal-Help
CBX6071P	PNLGRP	Analyze Audit Journal-Panel Group
CBX6071	RPGLE	Analyze Audit Journal-CPP 1
CBX6072	RPGLE	Analyze Audit Journal-CPP 2
CBX607X	CMD	Analyze Audit Journal
CBX607M	CLP	Analyze Audit Journal-Build command

To ease the process of creating the ANZAUDJRN command, I've included the CBX607M CL program. Simply compile and run the CBX607M program following the instructions in the source header, providing your target library as the only parameter.

Download the code bundle at SystemiNetwork.com/code.

-C.F.

Find Out More

["Forensic Analysis Using QAUDJRN, Part 1: Command Usage"](#)

September 2009, ID 63933

["Creating and Managing the QAUDJRN Journal"](#)

August 2006, ID 53121

["Common Sense Security Auditing"](#)

August 2004, ID 18842

"Extracting Information from QAUDJRN"

October 2008, ID 57332

"The Custom Command PRTPWDAUD (Print Password Audit Report)"

May 2009 e-newsletter, ID 58083

APIs by Example: Retrieve Journal Entries (QjoRetrieveJournalEntries)

ID 17008

Tip #2: IFS Journal Monitor:

October 2005, ID 20259

Retrieve Journal Entry Exit Program

ID 17415

"APIs at Work—with Jobs"

September 2006, ID 20661

"Writing RPG Code Then and Now"

October 2008, ID 62254

Source URL: <http://iprodeveloper.com/rpg-programming/analyze-your-audit-journal-rpg>