Getting Started With

APIs from RPG

Presented by

Scott Klement

http://www.scottklement.com

A programmer's wife tells him: "Run to the store and pick up a loaf of bread; If they have eggs, get a dozen." The programmer comes home with 12 loaves of bread.

What's an API?

API = Application Programming Interface

- An Interface

  APIs represent a way for one application to interface with another one. For example, Order Entry software might need to interface with Shipping software to determine a shipping charge.

- Program or Procedure Calls

  Usually APIs are implemented as programs or subprocedures that you call and pass parameters to.

- Program to Program (or Procedure to Procedure)

  APIs are designed to be used by programs. They're not (usually) intended to be called from the command line, menu, etc. Instead, they're called from a program. They don't take their input from a keyboard, but instead from a parameter. They write their output to a parameter, and not to a screen or paper. They are programs intended to be called by programs.

- Who writes APIs?

  Anyone can write an API. In fact, you've probably already written some.

The IBM i APIs

IBM provides over 3200 different programs and procedures that you can call to interface with the various functions of the operating system!

This presentation focuses on how to get started using the IBM i APIs from an RPG IV (ILE RPG) program.

We'll start by examining how IBM's documentation is laid out, and discuss how to find the API you're looking for, as well as which parameters it needs.

Methods for Finding APIs

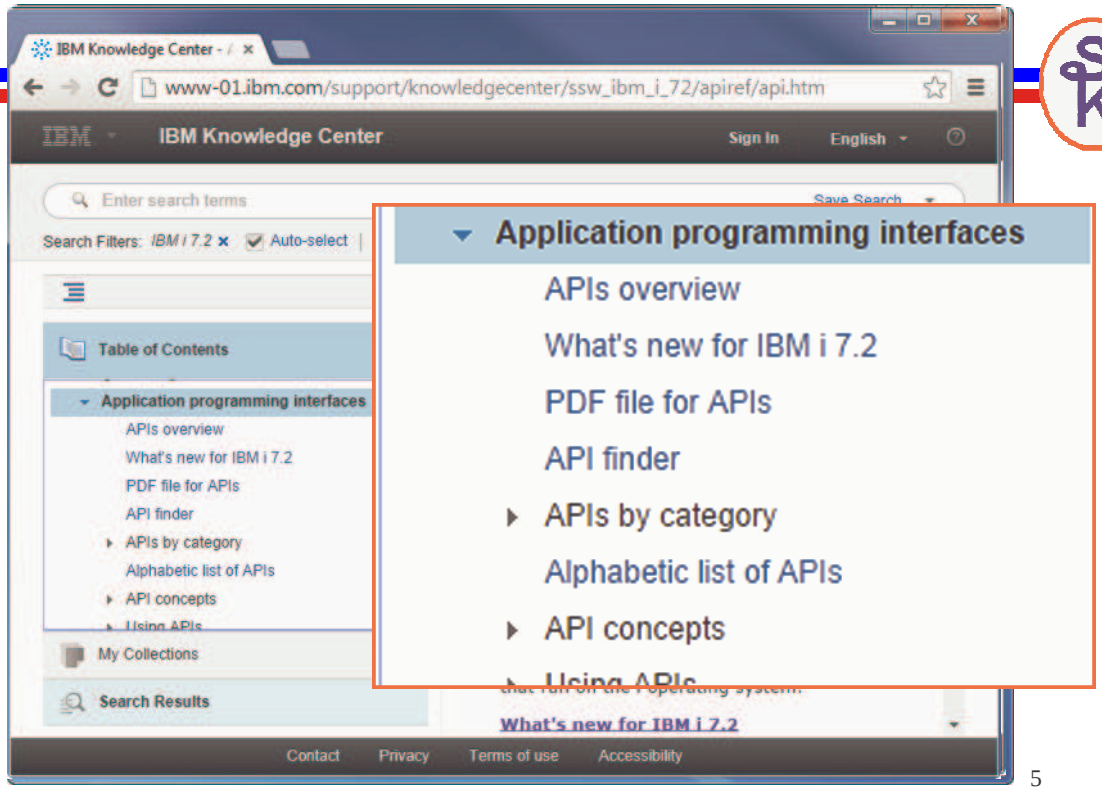
There are two different scenarios where you might be looking for information about APIs:

- When you know the name of the API, but you don't know what it does *(usually when you're trying to understand someone else's code)*
- When you know what you want to do, but you don't know which API does the job.

IBM provides 3 ways of finding APIs:

- APIs by Category *(When you don't know the API name.)*
- API finder *(When you do know the API name or title.)*
- Alphabetical Listing of APIs *(I've never found a use for this.)*

http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i/welcome



For example...

Let's say you're reading a program, and you see code like the following:

```
CHGVAR VAR(%BIN(&RCVVARLEN)) VALUE(1000)

CALL PGM(QDCRDEVD) PARM( &RCVVAR     +
                         &RCVVARLEN  +
                         'DEVD0600'  +
                         &DEV        +
                         &ERRCODE    )

CHGVAR VAR(&ADDR) VALUE(%SST(&RCVVAR 878 15))
```

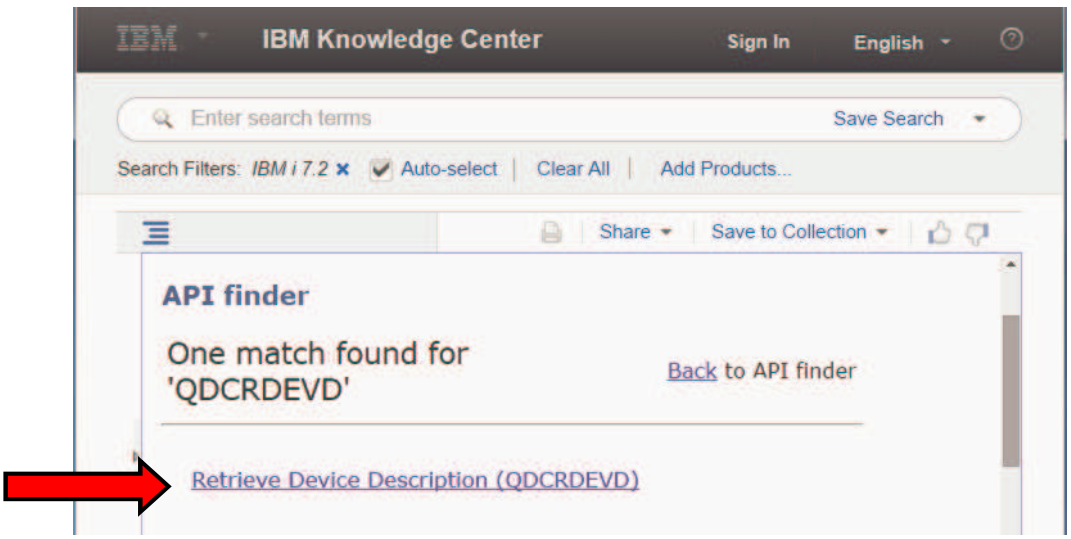
In this case, you may not be sure what QDCRDEVD does, but you know it's name. In that case, you want to be able to type the name and get information about the API.

To do that, you'll use the API Finder.

API Finder (1 of 2)

The API finder is for searching for an API. It's the "Google" for IBM i API information.

API Finder (2 of 2)



Found it… Now What?

Either the API finder has found a link to the API you were looking for, or you've found it by browsing the categories.

The next step is to click that link and read the documentation for the API itself.

This information is divided into sections that will be explained in upcoming slides:

- Parameter Summary Area
- API Description, Locks & Authority Info
- Detailed Information about parameters
- Error Information

API Parameter Summary

At the top of each API's page

Send Program Message (QMHSNDPM) API

QMHSNDPM is the name of the program to call.

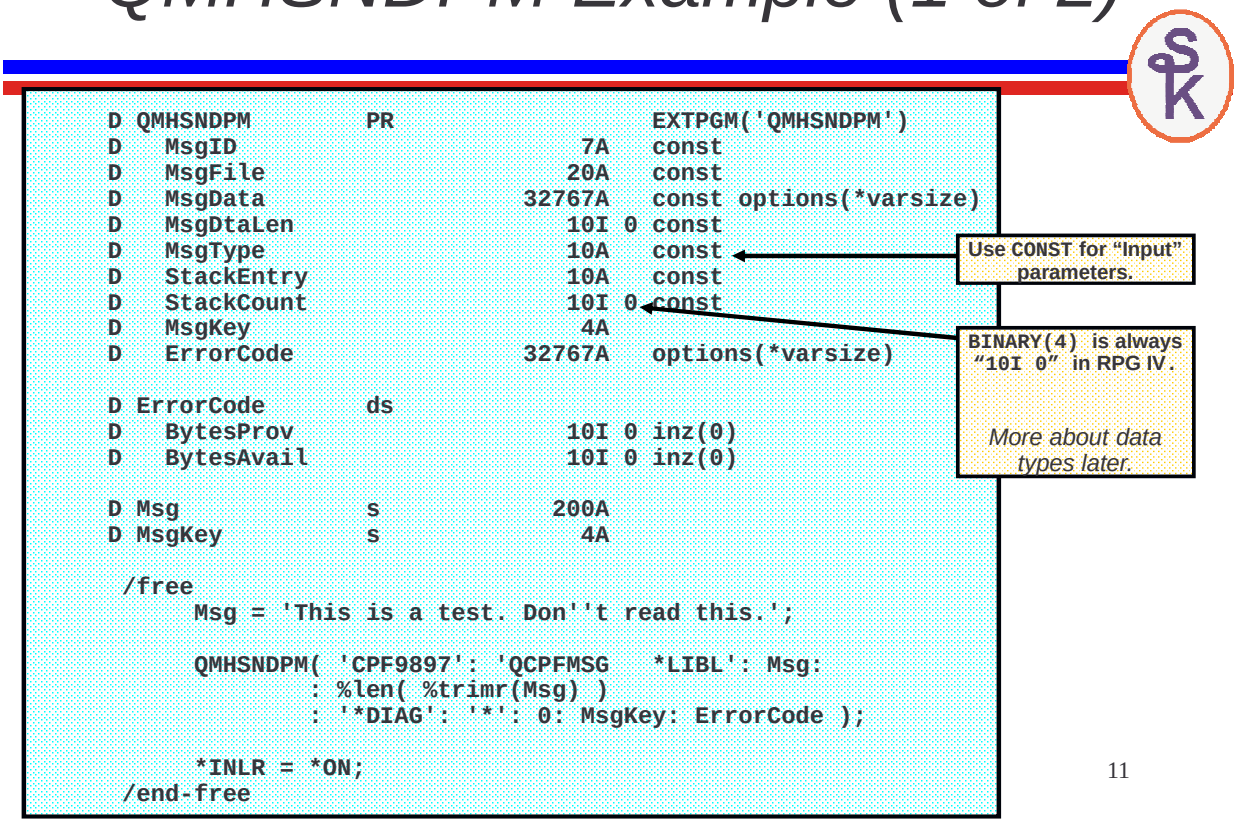
	Required Parameters Group:	How the API uses the parm:	Data type of the parm:
1	Message identifier	Input	Char(7)
2	Qualified message file name	Input	Char(20)
3	Message data or immediate text	Input	Char(*)
4	Length of message data or immediate text	Input	Binary(4)
5	Message type	Input	Char(10)
6	Call stack entry	Input	Char(*) or Pointer
7	Call stack counter	Input	Binary(4)
8	Message key	Output	Char(4)
9	Error code	I/O	Char(*)
	Optional Parameter Group 1:		
10	Length of call stack entry	Input	Binary(4)
11	Call stack entry qualification	Input	Char(20)
12	Display program messages screen wait time	Input	Binary(4)
	Optional Parameter Group 2:		
13	Call stack entry data type	Input	Char(10)
14	Coded character set identifier	Input	Binary(4)

These Parameters are Required, (must always be passed.)

These aren't required, but if you pass one, you have to pass them all.

Same here,PLUS in order to pass group 2, you must also pass group 1.

QMHSNDPM Example (1 of 2)



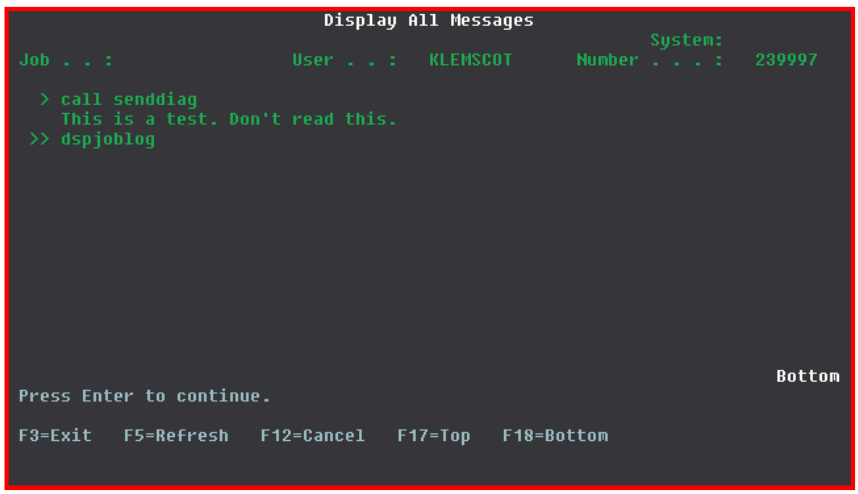
```
D QMHSNDPM        PR                  EXTPGM('QMHSNDPM')
D   MsgID                        7A   const
D   MsgFile                     20A   const
D   MsgData                  32767A   const options(*varsize)
D   MsgDtaLen                   10I 0 const
D   MsgType                     10A   const
D   StackEntry                  10A   const
D   StackCount                  10I 0 const
D   MsgKey                       4A
D   ErrorCode                32767A   options(*varsize)

D ErrorCode       ds
D   BytesProv                   10I 0 inz(0)
D   BytesAvail                  10I 0 inz(0)

D Msg             s              200A
D MsgKey          s                4A

 /free
      Msg = 'This is a test. Don''t read this.';

      QMHSNDPM( 'CPF9897': 'QCPFMSG   *LIBL': Msg:
              : %len( %trimr(Msg) )
              : '*DIAG': '*': 0: MsgKey: ErrorCode );

      *INLR = *ON;
 /end-free
```

Use CONST for "Input" parameters.

BINARY(4) is always "10I 0" in RPG IV.

More about data types later.

QMHSNDPM Example (2 of 2)



```
                          Display All Messages
                                                          System:
Job . . :                 User . . :   KLEMSCOT     Number . . . :   239997

  > call senddiag
    This is a test. Don't read this.
 >> dspjoblog

                                                                        Bottom
Press Enter to continue.

F3=Exit    F5=Refresh    F12=Cancel    F17=Top    F18=Bottom
```

Data Types

The data types that are listed for each API are *usually* pretty self explanatory.

Examples:

- CHAR(20) = character field, 20 long (**`20A`** in RPG)
- PACKED(15,5) = packed, 15 digits w/5 decimal places (**`15P 5`** in RPG)
- POINTER(SPP) = Pointer. (Data type **`*`** in RPG – more info later!)

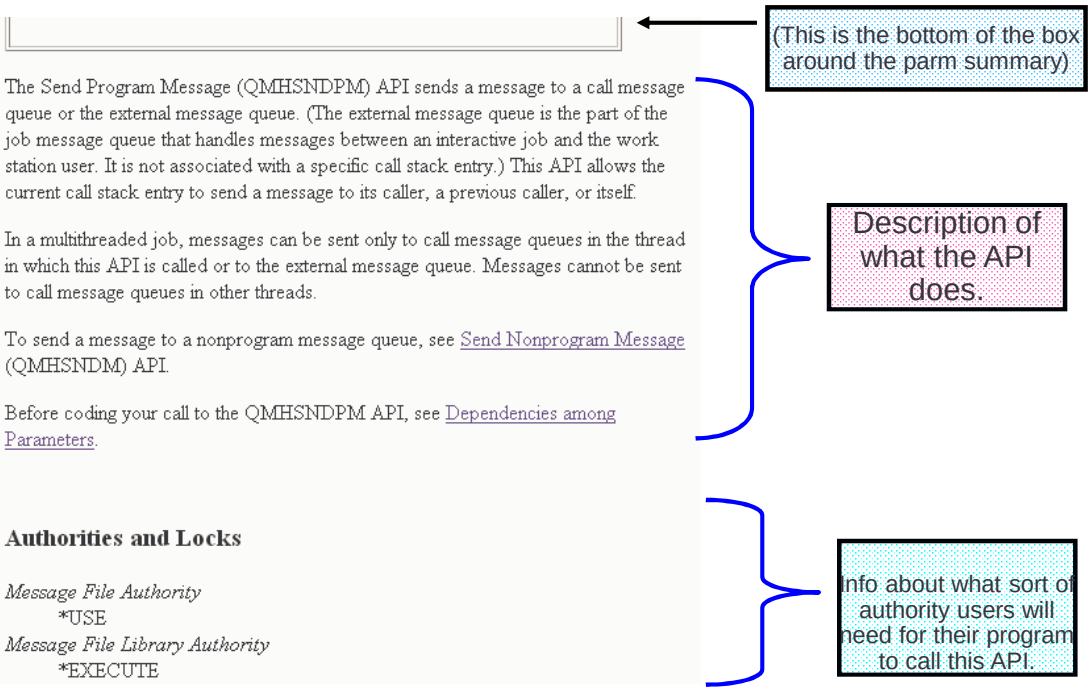
However, there are two data types that seem to cause a lot of confusion:

- BINARY(4) = 4 byte binary integer. (**`10I 0`** in RPG)
- BINARY(4), UNSIGNED = 4 byte unsigned binary integer (**`10U 0`** in RPG)
- CHAR(*) = Character field with a special length, *not VARYING* (Declare this as a long character field with **`options(*VARSIZE)`** on the prototype.)

NOTE: In RPG, we declare our numeric fields by the number of digits we can store in them. So a "9P 0" field is 5 bytes long, but stores a 9 digit number. A "10I 0" field is a binary integer that's 4 bytes long, but stores a 10 digit number. *NEVER USE THE "B" DATA TYPE, IT'S NOT A TRUE BINARY INTEGER. THE **I** AND **U** DATA TYPES ARE, AND THEY RUN MUCH FASTER, TOO.*

API Description

On the API's page, after the Parameter Summary.



Detailed Parameter Descriptions

On the API's page, after the Authorities and Locks

Required Parameter Group

Message identifier
INPUT; CHAR(7)

The identifying code for the predefined message being sent, or blanks for an immediate message.

When sending an escape, notify, or status message, you must specify a message identifier. When sending a request message, you must use blanks. When sending other types of messages, you can use either a message identifier or blanks.

If you specify a message identifier, you must specify a qualified message file name. If you do not specify a message identifier, the API ignores the qualified message file name parameter.

Qualified message file name
INPUT; CHAR(20)

For a predefined message, the name of the message file and the library in which it resides. The first 10 characters specify the file name, and the second 10 characters specify the library. You can use these special values for the library name:

**CURLIB* The job's current library

**LIBL* The library list

There are detailed descriptions of all of the APIs parameters.

This is what usually takes up most of the space on each API's page.

Errors the API can Return

At the end of each API's manual page

Error Messages

Message ID	Error Message Text
CPF24C5 E	Pointer to call stack entry not valid.
CPF24C6 E	Value of To call stack entry data type parameter not valid.
CPF24C8 E	Control boundary not found on call stack.
CPF24C9 E	Program boundary not found on call stack.
CPF24CB E	*PGMNAME requires a specified program name.
CPF24CC E	Call stack entry &2 for *PGMNAME not found.
CPF24CD E	Module name cannot be specified when *PGMBDY is used.
CPF24CE E	Qualifier &1 incorrect for use with pointer.
CPF24AC E	Either message identifier or message text must be specified.
CPF24AD E	Messages to remove must be *ALL if program message queue is *ALLINACT.
CPF24A3 E	Value for call stack counter parameter not valid.
CPF24BF E	Module or bound-program name is blank.
CPF24B3 E	Message type &1 not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF24B6 E	Length of &1, not valid for message text or data.
CPF24B7 E	Value &1 for call stack entry name length not valid.

Sometimes there are additional notes about why an error might be caused.

API Error Handling (1/2)

Offset		Use	Type	Field
Dec	Hex			
0	0	Input	Binary(4)	Bytes Provided
4	4	Output	Binary(4)	Bytes Available
8	8	Output	Char(7)	Exception ID
15	F	Output	Char(1)	Reserved
16	10	Output	Char(*)	Exception Data

- This structure is passed in a parameter to the API.
- *Bytes Provided* should tell the API how big the DS is. (That way, you can control the size of the Exception Data field!) *You must set this before calling the API. Don't leave it blank! (x'40404040' = 1,077,952,576)*
- *Bytes Available* tells you how much error data the API sent back.
- You can leave off the fields on the end, as long as Bytes Provided is correct.
- You can set *Bytes Provided* to zero if you'd like the API to send you an *ESCAPE message when it fails.

NOTE: The CEE APIs, and the Unix-type APIs have separate mechanisms for error handling that I do not cover here. They are documented in the Knowledge Center, however.

API Error Handling (2/2)

If you assume the API will always succeed do this. Then, if something weird does happen, the program will halt and there'll be good diagnostic info in the job log.

```
D ErrorCode       ds
D   BytesProv                 10I 0 inz(0)
D   BytesAvail                10I 0
```

Sends an *ESCAPE message. Program will crash if you don't monitor for it.

If you want to handle errors in your code, use this syntax instead. Nothing will go to the job log, it's up to you to handle errors:

```
D ErrorCode       ds
D   BytesProv                 10I 0 inz(%size(ErrorCode))
D   BytesAvail                10I 0 inz(0)
D   MsgId                      7A
D                              1A
D   MsgData                 1024A
     .
     .
  CALLP QMHSNDPM( ...other parms here... : ErrorCode);

  if ( BytesAvail > 0 );
     ErrMsg = MsgId + ' occurred called QMHSNDPM API!';
     // show ErrMsg to user!
  endif;
```

The use of %SIZE is a good idea. Let the compiler do the work, and help you when you need to make changes.

This way, if BytesAvail isn't zero after calling the API, you know there's an error.

Complex Parameters (Formats)

A format is a code that identifies the format of a data structure. (It's similar in concept to a record format.)

A format name typically looks something like this:

DEVD0600

When an API can return different types of data, or can return it in many different formats (or would like to be able to do that at some point in the future!) it requests a format.

Let's say you're writing an interactive program, and you want to know the IP address of your user's PC.

To find this out, you'll need to retrieve information about the Display Device that he's using. This is done with the "Retrieve Device Description (QDCRDEVD)" API.

This API returns all sorts of information about a device. There are hundreds of fields that it can return!

It returns different information, depending on what sort of device you'd like information about. A tape device (*TAP) has very different information than a display device (*DSP)!

Formats in the Manual (1/3)

Retrieve Device Description (QDCRDEVD) API

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Device name	Input	Char(10)
5	Error Code	I/O	Char(*)

Default Public Authority: *USE

Threadsafe: Yes

The Retrieve Device Description (QDCRDEVD) API retrieves information about a device description.

The first two parms tell the API which data structure to return info into.

The format name tells the API what the data structure looks like.

The "device name" tells the API which device you're interested in.

But, what do you put for the format name?

Formats in the Manual (2/3)

To find the possible format names, scroll down to the detailed information for the required parameter group. This is what you'll find:

Format name

INPUT; CHAR(8)

The content and format of the information returned for each device description. The possible format names are:

DEVD0100	Basic device information.
DEVD0200	Detailed information for device category *APPC
DEVD0300	Detailed information for device category *ASC
DEVD0400	Detailed information for device category *BSC
DEVD0500	Detailed information for device category *DKT
DEVD0600	Detailed information for device category *DSP
DEVD0700	Detailed information for device category *FNC
DEVD0800	Detailed information for device category *HOST
DEVD0900	Detailed information for device category *INTR
DEVD1000	Detailed information for device category *NET
DEVD1100	Detailed information for device category *PRT

Formats in the Manual (3/3)

To learn how your data structure must be formatted, scroll down to the detail info for DEVD0600

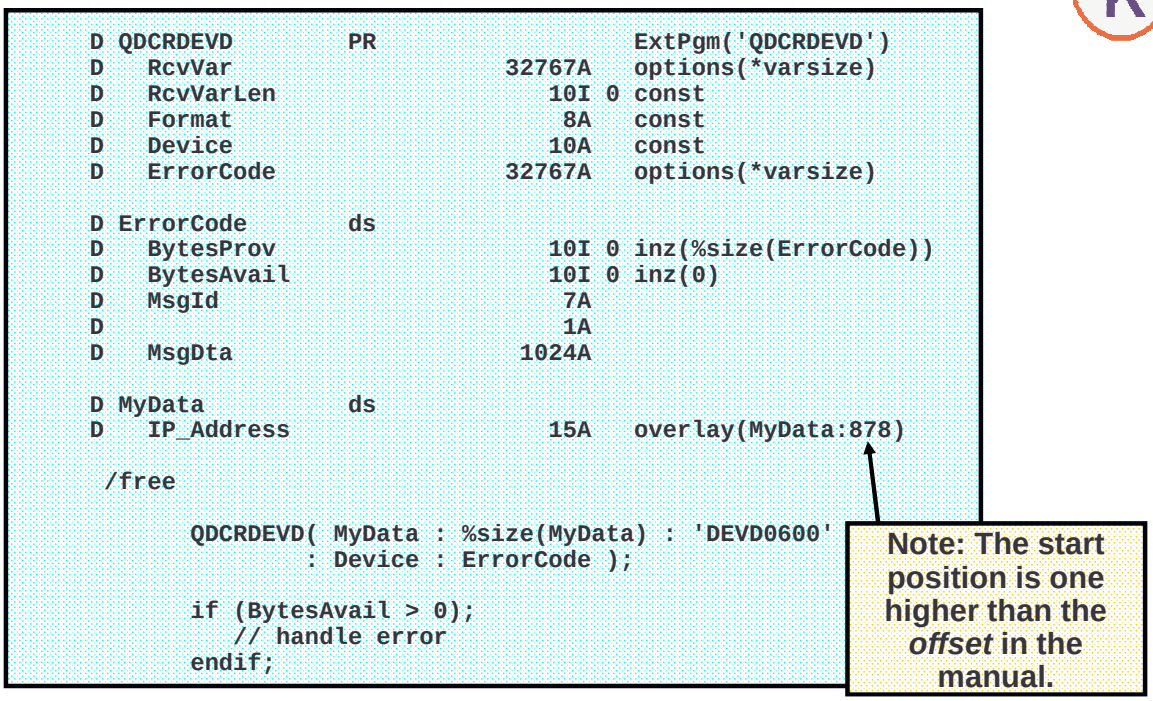
(part of it is shown on the right.)

DEVD0600 Format

This format returns detailed information about a device of category *DSP.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format DEVD0100
104	68	BINARY(4)	Character identifier: graphic character set
108	6C	BINARY(4)	Character identifier: code page
112	70	BINARY(4)	Maximum length of request unit
116	74	BINARY(4)	Inactivity timer
844	34C	BINARY(4)	Shared session number
848	350	CHAR(10)	Dependent location name
858	35A	CHAR(1)	Network protocol
859	35B	CHAR(18)	Network protocol address
877	36D	CHAR(15)	Internet Protocol (IP) internet address in dotted decimal form

Formats & Data Structures



```
D QDCRDEVD        PR                  ExtPgm('QDCRDEVD')
D   RcvVar                     32767A   options(*varsize)
D   RcvVarLen                    10I 0 const
D   Format                        8A   const
D   Device                       10A   const
D   ErrorCode                 32767A   options(*varsize)

D ErrorCode       ds
D   BytesProv                    10I 0 inz(%size(ErrorCode))
D   BytesAvail                   10I 0 inz(0)
D   MsgId                         7A
D                                 1A
D   MsgDta                     1024A

D MyData          ds
D   IP_Address                   15A   overlay(MyData:878)

 /free

       QDCRDEVD( MyData : %size(MyData) : 'DEVD0600'
               : Device : ErrorCode );

       if (BytesAvail > 0);
          // handle error
       endif;
```

Note: The start position is one higher than the *offset* in the manual.

Formats w/Variable Offsets

When the API docs tell you the position of the fields that it returns, it refers to that position as an *offset*.

OFFSET = Distance (in bytes) between the start of the data, and the point where the field starts.

In other words, it's a count of bytes from the start.
The first field is always at offset 0 because it's at the start.

Sometimes, the offset of data that it returns won't be at a fixed position. Instead, it'll pass you a variable that contains the offset of the field!

This is common when:

- Preceding data was variable-length.
- A list of repeating fields is returned. (such as a list of jobs on the system, list of objects in a library, etc.)

The best way to deal with variable offsets is with pointer logic.
Never, ever hard-code an offset when an API passes it to you in a parameter!

API Docs w/Var Offsets (1/2)

This is from format JOBI0750 of the Retrieve Job Information (QUSRJOBI) API.

It's for retrieving the library list for a given job.

62	3E	CHAR(2)	Reserved
64	40	BINARY(4)	Offset to libraries in system library list
68	44	BINARY(4)	Number of libraries in system library list
72	48	BINARY(4)	Offset to product libraries
76	4C	BINARY(4)	Number of product libraries
80	50	BINARY(4)	Offset to current library
84	54	BINARY(4)	Number of current libraries
88	58	BINARY(4)	Offset to libraries in user library list
92	5C	BINARY(4)	Number of libraries in user library list
96	60	BINARY(4)	Length of one library array entry
See note	See note	Array(*) of CHAR(*)	System library list (See Library array entry for format of library array entry.)
See note	See note	Array(*) of CHAR(*)	Product libraries (See Library array entry for format of library array entry.)
See note	See note	Array(*) of CHAR(*)	Current library (See Library array entry for format of library array entry.)
See note	See note	Array(*) of CHAR(*)	User library list (See Library array entry for format of library array entry.)

Note: The decimal and hexadecimal offsets depend on the number of libraries you have in the various parts of your library lists. The data is left-justified with a blank pad at the end. The array is sequential. It is an array or data structure. See CL Programming book for the total number of libraries that can be returned.

API Docs w/Var Offsets (2/2)

Each Library at the variable offsets follows the format of a "Library Array Entry". Here's that format:

Offset		Type	Field
Dec	Hex		
The fields repeat for each library object returned in the array.		CHAR(10)	Library name
		CHAR(50)	Library text description
		BINARY(4)	Library ASP number
		CHAR(10)	Library ASP name
		CHAR(*)	Reserved

Note that the length of that array entry is also variable.

The offset from the previous slide tells us where the first library is. The second library will be immediately after the first.

The *Length of One Library Array Entry* field tells us where the second one starts. (As well as the third, and fourth, and so on.)

Introduction to Pointers

The best way to handle variable offsets is with pointer logic.

POINTER = A variable that stores an address in the system's main storage (or, "memory").

Just as a packed (or zoned) field is a variable designed to hold a decimal number, and a date field is designed to hold a date, and a time field is designed to hold a time, a pointer field is designed to hold an address in your system's memory.

What can you do with pointer fields?

- Set them to *NULL (*"this pointer doesn't currently have an address in it."*)
- Store the address of another variable in them.
- Ask the system to reserve (or "allocate") memory, then store the address of that memory in them.
- Add an offset to them (calculate the address X number of bytes later in memory)
- Subtract one pointer from another to calculate the offset between them.
- Base a variable on them

Based Variables

- Memory isn't automatically reserved to store their values.
- Instead, you control the place in memory where they reside.
- You can change it on the fly by changing the pointer.

Trivial Pointer Examples

```
D FIELD1          s              10A
D p_Field2        s                *   inz(*NULL)
D FIELD2          s               1A   Based(p_Field2)
D FIELD3          s               7P 0 inz(1234567)

 /free
      Field1 = 'Mashville';

      p_Field2 = %addr(Field1);
      Field2 = 'N';
```

Field1 now contains "Nashville"

```
      p_Field2 = %addr(Field1) + 5;
```

Field2 now contains "i"

```
      Field1 = 'Strawbeary';
```

Field2 now contains "b"

```
      p_Field2 = p_Field2 + 2;
      Field2 = 'r';
      p_Field2 = %addr(Field3) + (%size(Field3) - 1);
      Field2 = x'0D';
```

Field1 now contains "Strawberry", Field3 contains -1234560

Hey wait, what happened to APIs?

62	3E	CHAR(2)	Reserved
64	40	BINARY(4)	Offset to libraries in system library list
68	44	BINARY(4)	Number of libraries in system library list
72	48	BINARY(4)	Offset to product libraries
76	4C	BINARY(4)	Number of product libraries
80	50	BINARY(4)	Offset to current library
84	54	BINARY(4)	Number of current libraries
88	58	BINARY(4)	Offset to libraries in user library list
92	5C	BINARY(4)	Number of libraries in user library list
96	60	BINARY(4)	Length of one library array entry
See note	See note	Array(*) of CHAR(*)	System library list (S format of library arra
See note	See note	Array(*) of CHAR(*)	Product libraries (Se format of library arra
See note	See note	Array(*) of CHAR(*)	Current library (See of library array entry
See note	See note	Array(*) of CHAR(*)	User library list (See of library array entry

Note: The decimal and hexadecimal offsets depend on t have in the various parts of your library lists. The data is left-justified with a blank pad at the end. The array is sequential. It is an array or data structure. See CL Programming book for the total number of libraries that can be returned.

Offset		Type	Field
Dec	Hex		
The fields repeat for each library object returned in the array.		CHAR(10)	Library name
		CHAR(50)	Library text description
		BINARY(4)	Library ASP number
		CHAR(10)	Library ASP name
		CHAR(*)	Reserved

So how do you read variable offsets returned by an API using pointers?

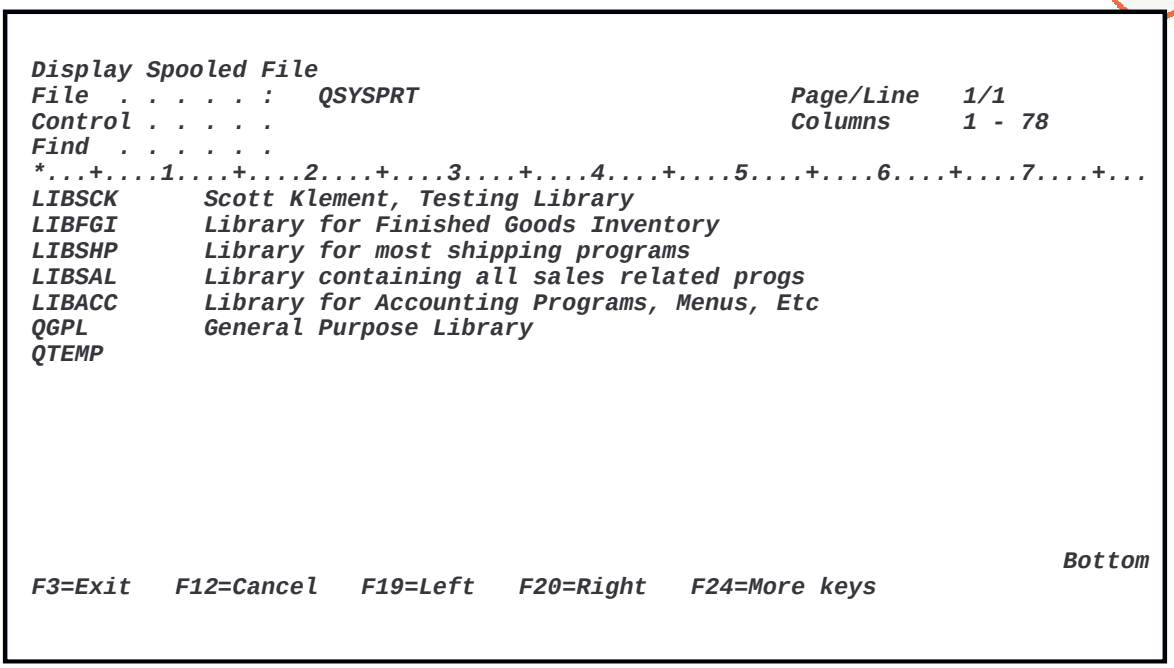
API Variable Offset Example (1/3)

```
FQSYSPRT   O    F   80        PRINTER

D QUSRJOBI        PR                  ExtPgm('QUSRJOBI')
D   RcvVar                     32767A   options(*varsize)
D   RcvVarLen                    10I 0 const
D   Format                         8A   const
D   QualJob                       26A   const
D   InternalId                    16A   const
D   ErrorCode                 32767A   options(*varsize: *nopass)
D   Reset                          1A   options(*nopass)

D MyData          ds                  based(p_MyData)
D  OffsetUsrLibl                  10I 0 overlay(MyData: 89)
D  NumUsrLibl                     10I 0 overlay(MyData: 93)
D  EntryLen                       10I 0 overlay(MyData: 97)

D LibEntry        ds                  based(p_LibEntry)
D  LibName                        10A
D  Text                           50A
D  ASPNo                          10I 0
D  AspName                        10A
```


API Variable Offset Example (2/3)

```
D DataSize        s                 10I 0
D x               s                 10I 0

 /free
      DataSize = 1024 * 1024;
      p_MyData = %alloc(DataSize);

      QUSRJOBI( MyData: DataSize: 'JOBI0750': '*': *blanks );

      for x = 0 to (NumUsrLibl - 1);
          p_LibEntry = p_MyData + OffsetUsrLibl + (x * EntryLen);
          except PrintLib;
      endfor;

      dealloc p_MyData;
      *inlr = *on;
 /end-free

OQSYSPRT   E           PrintLib
O                      LibName            10
O                      Text               62
```

API Variable Offset Example (3/3)



```
Display Spooled File
File  . . . . . :   QSYSPRT                          Page/Line   1/1
Control . . . . .                                    Columns     1 - 78
Find  . . . . . .
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+...
LIBSCK      Scott Klement, Testing Library
LIBFGI      Library for Finished Goods Inventory
LIBSHP      Library for most shipping programs
LIBSAL      Library containing all sales related progs
LIBACC      Library for Accounting Programs, Menus, Etc
QGPL        General Purpose Library
QTEMP
                                                                        Bottom
F3=Exit    F12=Cancel    F19=Left    F20=Right    F24=More keys
```

User Spaces

USER SPACE = A disk object that acts very much like a memory allocation.

Characteristics of a user space:

- Created by calling an API.
- Can be marked “auto-extend” so that they’ll automatically get bigger as needed. (With memory, you have to re-allocate to get a larger space.)
- You can get a pointer to a user space, and use it just as you would memory.
- You can base variables on a user space pointer, and use those variables like you would any other RPG variable.
- As a disk object, it can be saved in-between calls.
- Useful for remember “last time I ran this” values.
- It can be backed up to tape or optical media
- It can be shared with other jobs on the system.
- APIs exist for reading/writing user spaces for languages that don’t support pointers.
- That includes OPM languages.
- APIs that need to return data that might be too large for an HLL variable will put their data in a user space. That way, it’s accessible from any IBM i language.

List APIs

Many of the APIs that need to return a list of something (jobs, libraries, objects, modules, etc.) are called “List APIs”.

Characteristics:

- Accept a user space library/name to store the results in.
- The generated user space always starts with a “generic header”
- Generic header contains offset, count and entry size information needed to read the list.
- The format of the list entries will vary depending on the API.

For example, you might want to get a list of the interactive jobs that are active on the system. So you’d look for an API that does that.

- APIs by Category
- Work Management (deals with how the system processes it’s workload)
- List Jobs (QUSLJOB) sounds good!

List API Example (1/4)

```
FQSYSPRT   O    F   80        PRINTER

D QUSCRTUS        PR                   ExtPgm('QUSCRTUS')
D   UserSpace                    20A   CONST
D   ExtAttrib                    10A   CONST
D   InitialSize                  10I 0 CONST
D   InitialVal                    1A   CONST
D   PublicAuth                   10A   CONST
D   Text                         50A   CONST
D   Replace                      10A   CONST options(*nopass)
D   ErrorCode                 32767A   options(*varsize:*nopass)

D QUSPTRUS        PR                   ExtPgm('QUSPTRUS')
D   UserSpace                    20A   CONST
D   Pointer                        *

D QUSDLTUS        PR                   ExtPgm('QUSDLTUS')
D   UserSpace                    20A   CONST
D   ErrorCode                 32767A   options(*varsize)
```

API to create a user space.

API to get a pointer to a user space.

API to delete a user space object (when we’re done.)

List API Example (2/4)

```
D QUSLJOB         PR                  ExtPgm('QUSLJOB')
D   UserSpace                   20A   CONST
D   Format                       8A   CONST
D   QualJob                     26A   CONST
D   Status                      10A   CONST
D   ErrorCode                32767A   options(*varsize:*nopass)

D ErrorCode       ds
D   BytesProv                  10I 0 inz(0)
D   BytesAvail                 10I 0 inz(0)

D ListHeader      ds                  based(p_ListHeader)
d   ListOffset                 10I 0 overlay(ListHeader:125)
d   EntryCount                 10I 0 overlay(ListHeader:133)
d   EntrySize                  10I 0 overlay(ListHeader:137)

D Entry           ds                  based(p_Entry)
D   JobName                     10A
D   JobUser                     10A
D   JobNbr                       6A
D   IntJobId                    16A
D   Status                      10A
D   Type                         1A
D   SubType                      1A

D x               s             10I 0
D offset          s             10I 0
```

API to list jobs into a user space.

API Error Code

Generic List Header DS

Data structure for format JOBL0100

List API Example (3/4)

```
/free
     QUSCRTUS( 'JOBLIST   QTEMP'
             : 'LISTAPI'
             : 1024 * 1024
             : x'00'
             : '*EXCLUDE'
             : 'User Space to Contain Job List'
             : '*YES'
             : ErrorCode );

     QUSLJOB( 'JOBLIST   QTEMP'
            : 'JOBL0100'
            : '*ALL      *ALL      *ALL'
            : '*ACTIVE'
            : ErrorCode );

     QUSPTRUS( 'JOBLIST   QTEMP'
             : p_ListHeader );

     for x = 1 to EntryCount;

         offset = ListOffset + (x-1) * EntrySize;
         p_Entry = p_ListHeader + offset;

         if (Type = 'I');
            except;
         endif;
     endfor;

     QUSDLTUS('JOBLIST   QTEMP'
             : ErrorCode );
     *INLR = *ON;
/end-free
```

Create a User Space called QTEMP/JOBLIST, that's 1mb long.

List all active jobs to user space.

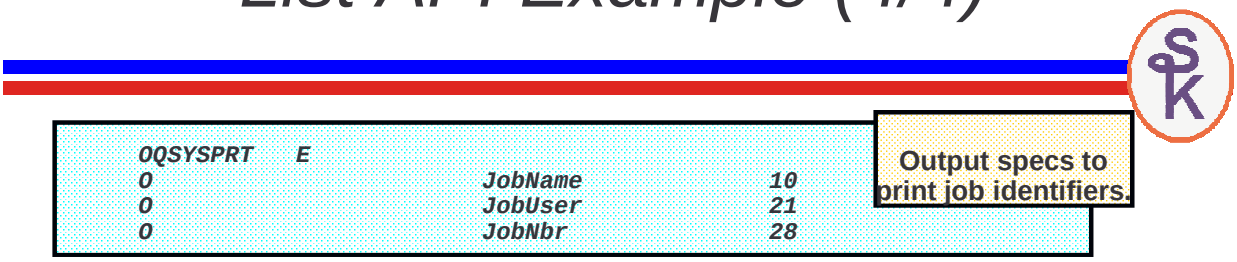
Get a pointer to the user space.

Calculate the offset to each entry, and point the ENTRY data structure at it.

If it's an interactive job, print it out.

Delete the user space and end the program.

List API Example (4/4)



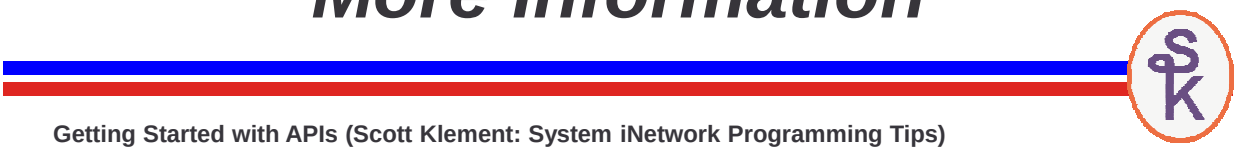
```
     OQSYSPRT   E
     O                       JobName          10
     O                       JobUser          21
     O                       JobNbr           28
```

Output specs to print job identifiers.

Drumroll please… and the results are….

```
Display Spooled File
File  . . . . . :   QSYSPRT                          Page/Line   1/1
Control . . . . .                                    Columns     1 - 78
Find  . . . . . .
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+...
QPADEV0001 BIZUJAME   239996
DSP01      KLEMSCOT   241320
ROGER      KLEMROGE   242304
SYSCON     QSECOFR    242331
DSP07      MARYZ      242326
S9S1       CHERYL     242223
```


More Information



Getting Started with APIs (Scott Klement: System iNetwork Programming Tips)
http://iprodeveloper.com/rpg-programming/getting-started-apis-0

Getting Started with APIs, Part 2
http://iprodeveloper.com/rpg-programming/getting-started-apis-part-2

Getting Started with APIs. Follow up to Part 2
http://iprodeveloper.com/rpg-programming/follow-getting-started-apis-part-2

Getting Started with APIs, Part 3
http://iprodeveloper.com/rpg-programming/getting-started-apis-part-3-0

Getting Started with APIs, Part 4
http://iprodeveloper.com/rpg-programming/getting-started-apis-part-4

APIs by Example (Carsten Flensburg)
http://iprodeveloper.com/search/results/APIs%20By%20Example

Fun with Pointers (Scott Klement: Personal Web site):
http://www.scottklement.com/rpg/pointers.html

IBM i Knowledge Center:
http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i/welcome

This Presentation

You can download a PDF copy of this presentation from:

http://www.scottklement.com/presentations/

Thank you!