

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

APIs by Example: Identifying and Working with Service Program References

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 09/25/2008 (All day)

The concept of service programs is one of the most valuable additions of all time to the RPG application development toolset on the System i. The ability to externalize and group specific and well-defined program functions into service programs makes it so much easier to support function reusability as well as focus on the core program logic and functionality in your programs. Perhaps even more than a practical achievement, service programs offer a welcome twist to a programmer's mindset, in terms of the design and development approach promoted and encouraged when taking advantage of service programs in your daily programming efforts.

Organizing and maintaining service programs, however, requires care and thought. The system supports and enforces integrity between service programs and the programs or service programs to which they're bound. Because the binding to a service program and its exported subprocedures or data is established at the point where a referencing program or service program is created, the system attempts to verify at program activation time that the service program has not changed in an incompatible way since the initial binding. Sometimes the ability to resolve service programs' references and verify maintained integrity ahead of time therefore comes in handy. And so do APIs.

To ensure integrity between a service program and the programs or service programs referencing it, all service programs are tagged with a current signature. This signature identifies the number and sequence of the exported procedures and data from the service program and is defined whenever the service program is created. You have the option of letting the system generate the signature by specifying the keyword EXPORT(*ALL) on the Create Service Program (CRTSRVPGM) command. This will cause the compiler to generate a signature based on the number, names, and order of the service program's subprocedures being defined with the EXPORT keyword on their procedure interface specifications.

If you want to control service program signatures yourself you have the option of using binder language. Binder language also allows you to explicitly specify which of the exported subprocedures you actually want to make available to other programs and service programs as well as to specify a signature of up to 16 bytes on the binder language's Start Program Export List (STRPGMEXP) command, as in the following example:

```
StrPgmExp  PgmLvl( *CURRENT )  Signature( 'UsrAppInf_01.0.0' )

Export symbol( "DLTUSRTIMZON" )
Export symbol( "GETUSRTIMZON" )
Export symbol( "SETUSRTIMZON" )
Export symbol( "VFYUSRTIMZON" )
```

```
EndPgmExp
```

You can still have the compiler generate the signature by specifying the keyword `SIGNATURE (*GEN)` though, but I recommend that you specify the signature yourself to ensure a human-readable and reproducible result. By adding new subprocedures to the end of the export list, binder language also allows you to maintain and support previous versions of the service program by specifying a program level (PGMLVL) of `*PRV` for blocks of subprocedures supported by a previous version of the service program. A service program therefore optionally supports more than one signature. By specifying the signature yourself, you also have the option of reusing the signature for future versions, as long as any new exported subprocedure is added to the end of the export list.

This way you can continue to add procedures to a service program without causing signature violations to occur to programs bound to earlier versions of the service program. Note that in order to start taking advantage of binder language for existing service programs, you can generate binder language members for these by using the Retrieve Binder Source (RTVBNDSRC) command. You then have the option of adapting the binder source and subsequently re-creating the service program, specifying the binder language member as the export source on either the CRTSRVPGM or UPDSRVPGM (Update Service Program) command.

To look up signature information for a specific service program, use the command Display Service Command (DSPSRVPGM) specifying the keyword `DETAIL(*SIGNATURE)`:

```
DSPSRVPGM SRVPGM(CBX001) DETAIL(*SIGNATURE)
```

For the specified program, the above command leads to the panel below, after pressing function key F11 to see the character version of the signature instead of the hexadecimal format:

Display Service Program Information	
Display 1 of 1	
Service program	: CBX001
Library	: QGPL
Owner	: CARSTEN
Service program attribute	: RPGLE
Detail	: *SIGNATURE
Signatures:	
UsrAppInf_01.5.0	
UsrAppInf_01.2.0	

```

Bottom
F3=Exit    F11=Display hexadecimal signature    F12=Cancel    F17=Top

F18=Bottom

```

The above service program CBX001 supports two program signature levels, UstrAppInf_01.5.0, which is the current level, and UstrAppInf_01.2.0, which is a previous level. As I will explain in the following section, any program or service program referencing the CBX001 service program must store one of the two signatures to be allowed to successfully activate the CBX001 service program.

Here's how it goes: When a program or service program referencing one or more subprocedures in a (nother) service program is created, the binding process performed during program creation will link the referenced service program to the program being created. This binding process will retrieve and store the current signature of the service program into the program or service program being created.

When later the program or service program is activated, the system will attempt to check and verify that any referenced service program still supports the stored signature. If this check fails, the activation process will terminate, and an exception message will be sent to the caller. This facility is conceptually very much like the record-level identifier used to verify that externally defined files have not changed since program compilation time, whenever a program referencing the file is activated, and generates a level check exception if a change is detected.

To examine what signature level a program or service program requires to successfully activate a specific service program, you can use the Display Program (DSPPGM) and Display Service Program (DSPSRVPGM) commands, respectively, specifying the DETAIL(*SRVPGM) keyword:

```
DSPPGM PGM(CBX101) DETAIL(*SRVPGM)
```

Again, you'll need to press F11 to see the character version of the signatures:

```

                                Display Program Information

Display 1 of 1
  Program . . . . . : CBX101      Library . . . . . :
QGPL
  Owner . . . . . : CARSTEN

  Program attribute . . : RPGLE

  Detail . . . . . : *SRVPGM

Type options, press Enter.

5=Display

```

Service			
Opt	Program	Library	Signature
	CBX001	*LIBL	UsrAppInf_01.5.0
	QRNXIE	QSYS	QRNXIE
	QRNXUTIL	QSYS	QRNXUTIL
	QLEAWI	QSYS	à7 ¿µee¬#ÿ¬1n8¬A
Bottom			
F3=Exit	F4=Prompt	F11=Display hexadecimal signature	
F12=Cancel			
F17=Top	F18=Bottom		

If you're interested in reading more about the concept of service program signatures, be sure to use the links I've included at the end of this article. They reference articles that discuss this topic thoroughly

Anyway, at this point it is obviously possible to establish a cross reference between service programs and the programs and service programs bound to it. It just requires a lot of manual work to inspect all the relevant candidates for a reference to the service program in question, followed by a visual verification of the individual signature levels.

If you therefore arrive at a situation in which you'll need to change a signature, remove a subprocedure, reorganize and split up an extending service program, or something similarly incompatible with maintaining a supported signature for existing programs or service programs--or if you simply want to document and establish a cross reference report for specific service programs--you could be looking at a challenging and time-consuming task.

That was the conclusion I arrived at when placed in a similar situation recently. So I decided to instead spend the time building a tool allowing me to let the system do the hard work. The Work with Service Program References (WRKSPGREF) command was the result of this effort, here's what the command's prompt looks like:

Work with Service Program Ref (WRKSPGREF)			
Type choices, press Enter.			
Service program	Name	
Library	*LIBL	Name, *LIBL,
*CURLIB			
Reference program	Name, generic*,	

```

*ALL
  Library . . . . . *LIBL      Name, *LIBL,
*CURLIB...
  Reference program type . . . . . *ANY      *ANY, *PGM,
*SRVPGM
  Sort order . . . . . *OBJLIB      *OBJLIB, *TYPOBJ,
*LIBOBJ...
  Output . . . . . *              *, *PRINT

```

You specify the service program whose references you want to identify and signature level check as the primary parameter. Next you enter the generic name and library qualification of the relevant selection of programs and service programs to retrieve and check. Using the special name value `*ALL` and one of the special values available for library qualification you can potentially list and examine a lot of programs and service programs. This could of course lead to an extensive use of system resources, so if possible, narrow the field of candidates appropriately.

You also have the option of selecting only one type of programs for scrutiny as well as specify a variety of sort orders for the produced list. Finally you can choose to print the program list instead of displaying a work with panel. The available online help text offers more detail about the command and its parameters.

Here's a list of the core APIs and steps involved in producing a list of programs and service programs bound to a specific service program:

1. Call the Retrieve Service Program Information (QBNRSPGM) API to retrieve the current signature of the specified service program.
2. Call the List Service Program Information (QBNLSPGM) API to retrieve a list of all signatures currently supported by the specified service program.
3. For each program and service program identified by the command REFPGM selection criteria list all bound service programs using the List ILE Program Information (QBNLPGMI) API and List Service Program Information (QBNLSPGM) API, respectively.
4. For all programs and service programs referencing the service program in question match the signature against:

- a. The current signature of the service program
- b. The list of all signatures supported by the service program.

5. If a) produces a match the referencing program or service program is at a current signature level with the service program.
6. If b) produces a match the referencing program or service program is at a previous signature level with the service program (aka back-level).
7. If neither a) nor b) produces a match the referencing program or service program will generate a signature violation (MCH4431) upon an attempted initiation of the program or service program.

To show you an example of the WRKSPGREF display panel and to visualize the effect of the above algorithm, I ran the following command on my system:

```
WRKSPGREF SRVPGM(QGPL/CBX001)
          REFPGM(QGPL/CBX*)
          REFPGMTYP(*ANY)
          ORDER(*OBJLIB)
          OUTPUT(*)
```

The command returned the following program list display:

Work with Service Program References

WYNDHAMW

17-09-08

12:21:23

Service program : CBX001

Library : QGPL

Current signature . . . : UsrAppInf_01.5.0

Type options, press Enter.

2=Update 4=Delete 5=Display 6=Print 7=Rename 8=Program reference

Program

Reference

Reference

Signature

Opt Name Library Type Library Signature

State

CBX101 QGPL *PGM *LIBL UsrAppInf_01.5.0

*CURRENT

CBX102 QGPL *PGM *LIBL UsrAppInf_01.2.0

*BACKLEVEL

CBX103 QGPL *PGM *LIBL UsrAppInf_01.0.0

*SIGVIOL

Bottom

Parameters or command

```
===>
```

```
F3=Exit    F4=Prompt    F5=Refresh    F8=Display service program
F9=Retrieve
F11=Display hexadecimal    F12=Cancel    F17=Top    F18=Bottom
```

Using the function key F11 you can toggle between a hexadecimal signature format, a character signature format, and a program description. Function key F8 allows you to run the Display Service Program (DSPSRVPGM) command against the specified service program. A number of list options are included to execute various commands against the found programs and service programs. The list of available commands includes UPDPGM/UPDSRVPGM, DLTPGM/DLTSRVPGM, DSPPGM/DSPSRVPGM, RNMOBJ, and DSPPGMREF. Again, you also have online help text available to explain the list panel, columns, options and function keys.

This APIs by Example includes the following sources:

```
CBX196  -- RPGLE  -- Work with Service Program References - CPP

CBX196E -- RPGLE  -- Work with Service Program References - UIM Exit
Program
CBX196H -- PNLGRP -- Work with Service Program References - Help
CBX196P -- PNLGRP -- Work with Service Program References - Panel
Group
CBX196V -- RPGLE  -- Work with Service Program References - VCP

CBX196X -- CMD    -- Work with Service Program References

CBX196M -- CLP    -- Work with Service Program References - Build
Command
```

To create all the above objects, compile and run CBX196M, following the instructions in the source header. As always, you'll also find compilation instructions in the respective source headers.

Previously published articles explaining the concept of service program exports:

Barbara Morris: Maintainable Service Programs

<http://systeminetwork.com/article/maintainable-service-programs>

Scott Klement: Make a List of Your Exports

<http://systeminetwork.com/node/61306>

Simon Coulter: Service Program Signature Violations (Midrange Wiki)

http://wiki.midrange.com/index.php/Service_Program_Signature_Violations

This article demonstrates the following Program and CL Command APIs:

Retrieve Program Information (QCLRPBMI) API:

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

Retrieve Service Program Information (QBNRSPGM) API:

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

List ILE Program Information (QBNLPGMI) API:

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

List Service Program Information (QBNLSPGM) API:

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

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

http://www.pentontech.com/IBMContent/Documents/article/57221_669_WrkSpgRef.zip.

Source URL: <http://iprodeveloper.com/rpg-programming/apis-example-identifying-and-working-service-program-references>