

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

## APIs by Example: Locating and Working with Module Imports

[System iNetwork Programming Tips Newsletter](#)

[Carsten Flensburg](#)

Carsten Flensburg

Thu, 10/23/2008 (All day)

In response to the [Work with Service Program References \(WRKSPGREF\) command](#) that I provided in the September 25, 2008, installment of APIs by Example, I received a note from a reader asking for a similar tool that finds the programs or service programs that import a given procedure. Such a tool would be helpful if you need to alter the procedure interface in a way that would be incompatible with the current interface or if you want to perform an impact analysis given a requirement to remove or replace a procedure.

I welcome suggestions or ideas for new API-driven commands and utilities! When I receive a suggestion based on actual business need, I'm inspired to try to come up with a new API example. So I quickly envisioned a new Work with Program Import (WRKPGMIMP) command and embarked on the initial steps involved in the design of such a utility.

However, during my initial research, I realized that no direct way exists to examine the program or service program object and identify the procedures a module imports. However, if the module object still exists and remains unchanged, it is possible to establish the required reference between program and procedure.

The requirement of keeping the module around and intact potentially reduces the usefulness of the WRKPGMIMP command. For example, programs created by using a Create Bound (CRTBNDxxx) command, such as CRTBNDRPG, will do their the binding against a temporary module object in QTEMP, and therefore there's no module object when you want to run WRKPGMIMP. So I added an option to the WRKPGMIMP command to help you locate all programs and service programs that contain modules that no longer exist or modules that are out of date (i.e., the source has been changed since the module was created). To take advantage of this option, you must specify the special value \*VFYMODREF for the WRKPGMIMP command's IMPORT parameter.

But let me start out with a brief discussion of the module anatomy of ILE programs and service programs. Both program types can be created from one or more modules all together making up the final program object. For programs, one module will provide the program entry module, and this is the one specified on the Create Program (CRTPGM) command's Entry module (ENTMOD) parameter. This parameter by default points to the \*FIRST module specified for the command's module list (MODULE) parameter. For service programs, no single entry point exists, as each exported subprocedure effectively constitutes an individual entry point.

To see the name of the entry module for a program object named CBX001 in library QGPL, run the command:

```
DSPPGM PGM(QGPL/CBX001) DETAIL(*BASIC)
```

To see all subprocedures and data items exported from a service program named CBX101 in library QGPL, run the command:

```
DSPSRVPGM SRVPGM(QGPL/CBX101) DETAIL(*PROCEXP)
```

A detailed discussion of how to control a service program's exports through binder language was provided in the previous APIs by Example. I've provided a link to that article at the end of this article.

Just as service programs can make exported procedures available to other programs or service programs, so can a module. All subprocedures in a module specifying the procedure interface keyword EXPORT can be resolved by the binding process when a program or service program is created and the module in question is specified on the create command's MODULE parameter. To see what procedures and data items are exported from a module named CBX001 in library QGPL, run the command:

```
DSPMOD MODULE(QGPL/CBX001) DETAIL(*EXPORT)
```

Likewise, all modules also have the option of importing subprocedures or data items that are employed by the module. When the module is created, all subprocedures or data items not resolved from the module itself are listed as part of the module's imported unresolved symbols array. These imports must be resolved at the point at which the modules are bound into a program or service program, unless OPTION(\*UNRSLVREF) is specified on the Create Program command. To see what unresolved import symbols are defined for a module object named CBX001 in library QGPL, run the command:

```
DSPMOD MODULE(QGPL/CBX001) DETAIL(*IMPORT)
```

The program binding process examines all modules' unresolved imports and attempts to resolve these by checking all modules and service programs submitted by the program creation command and specified explicitly by qualified name or implicitly through binding directories. As for the latter, some system binding directories are automatically present through the compiler, whereas other binding directories can be specified on the Create Program command or in the header specifications (H-spec) in the individual modules. To see what binding modules apply to a module named CBX001 in library QGPL, run the command:

```
DSPMOD MODULE(QGPL/CBX001) DETAIL(*REFSYSOBJ)
```

You'll see a list of binding directories, either included by the compiler or defined in the module's H-specification. Finally, and this is as close as we get to imported procedures by means of the program object, the program object will define all service program references resolved during the program binding process. All resolved imports will be found in either the program's module listing or the program's service program listing. To see what modules are bound into a program object named CBX001 in library QGPL, run the command:

```
DSPPGM PGM(QGPL/CBX001) DETAIL(*MODULE)
```

And to see what service programs are referenced, in terms of imports resolved from each listed service program, for the same program, run the command:

```
DSPPGM PGM(QGPL/CBX001) DETAIL(*SRVPGM)
```

However, although this command tells you which modules and service programs a given program uses, it does not tell you which procedures are called in those service programs or modules. For

example, if I find myself in a situation in which I need to uncover which programs or service programs import a specific subprocedure, I will be able to do so only by using the original module object as a bridge between the programs and the procedure. In other words; I will need to employ a number of APIs to perform the following lookup process:

1. Call the Open List of Objects API (QGYOLOBJ) to list all programs and service programs identified by the WRKPGMIMP command's REFPGM parameter.
2. For each program and service program found, list all modules bound into the program or service program by using the List ILE Program Information (QBNLPGMI) API and the List Service Program Information (QBNLSPGM) API, respectively.
3. For each bound module, perform the following verification and investigation process:

```

a. Check the module object existence using the Retrieve Object
   Description (QUSROBJD) API.
b. If a) is passed, verify the module source level against the
   corresponding data recorded
      into the program object by using the Retrieve Module
   Information (QBNRMODI) API.
c. If b) is passed, retrieve and process the module import
   symbol list by using the List
      Module Information (QBNLMODI) API and match each import
   symbol name and type against
      the symbol name and type specified as the WRKPGMIMP command's
   IMPORT parameter.

```

4. If c) produces a match, program and module information is included in the WRKPGMIMP command's work-with list.

Note, that if the aforementioned value \*VFYMODREF is specified for the WRKPGMIMP command's IMPORT parameter, only modules failing either test a) or b) are included in the work-with list.

Here's the WRKPGMIMP command's prompt panel:

Work with Program Import (WRKPGMIMP)

Type choices, press Enter.

Imported symbol name . . . . .		
Import symbol type . . . . .	*PROC	*PROC, *DATAITEM
Import program . . . . .		Name, generic*,
*ALL		
Library . . . . .	*LIBL	Name, *LIBL,
*CURLIB...		
Import program type . . . . .	*ANY	*ANY, *PGM,
*SRVPGM		

```

Sort order . . . . . *OBJLIB      *OBJLIB, *TYPEOBJ,
*LIBOBJ...
Output . . . . . *      *, *PRINT

```

You specify the name of the procedure or data item to which you want to find all program references as the primary parameter and the symbol type as the secondary. Please note that the symbol name is case sensitive, as are import symbol names. Next, you enter the generic name and library qualification of the relevant selection of programs and service programs whose modules you want to list and check for unresolved imports. 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 selection range as much as feasible.

You can also specify only one program type to be included in the array of programs to examine 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. When I run the following command on my system:

```

WRKPGMIMP IMPORT (GETSYSVAL)
          SYMTYP (*PROC)
          IMPPGM (QGPL/CBX*)
          IMPPGMTYP (*ANY)
          ORDER (*OBJLIB)
          OUTPUT (*)

```

I'm presented with the following work-with panel:

```

                                Work with Program Import

WYNDHAMW                                18-10-08
15:31:15
Import symbol . . . . . :   GETSYSVAL

Symbol type . . . . . :   *PROC

Type options, press Enter.

  2=Update           4=Delete program   5=Display program   6=Display
module
  7=Work with PDM    8=Program reference 9=Module reference

Program                                Module                                Module
Opt  Name      Library  Type  Name      Library
Status

```

```

        CBX101      QGPL      *PGM      CBX101      QGPL
*IMPSYMFND
        CBX102      QGPL      *PGM      CBX101      QGPL
*IMPSYMFND
        CBX103      QGPL      *PGM      CBX101      QGPL
*IMPSYMFND
        CBX110      QGPL      *PGM      CBX110      QGPL
*IMPSYMFND
        CBX123      QGPL      *PGM      CBX123      QGPL
*IMPSYMFND
        CBX125      QGPL      *PGM      CBX125      QGPL
*IMPSYMFND
        CBX130      QGPL      *PGM      CBX130      QGPL
*IMPSYMFND

More...
Parameters or command

===>

F3=Exit      F4=Prompt      F5=Refresh      F9=Retrieve      F11=Display
program text
F12=Cancel    F17=Top      F18=Bottom    F22=Display entire import
symbol

```

Using the function key F11, you can toggle between different types of program and module information. A number of list options are included to execute various commands against the found programs and service programs. The list of available commands includes UPDPPGM/UPDSRVPGM, DLTPGM/DLTSRVPGM, DSPPGM/DSPSRVPGM, DSPMOD, WRKMBRPDM, and DSPPGMREF. Again, you have online help text available to explain the list panel, columns, options, and function keys.

### This APIs by Example includes the following sources:

```

CBX197  -- RPGLE  -- Work with Program Import - CPP
CBX197E -- RPGLE  -- Work with Program Import - UIM Exit
CBX197H -- PNLGRP -- Work with Program Import - Help
CBX197P -- PNLGRP -- Work with Program Import - Panel Group
CBX197V -- RPGLE  -- Work with Program Import - VCP
CBX197X -- CMD    -- Work with Program Import

CBX197M -- CLP    -- Work with Program Import - Build Command

```

To create all these objects, compile and run CBX197M, following the instructions in the source header. As always, you'll also find compilation instructions in the respective source headers.

### Previously published related APIs by Example article:

APIs by Example: Identifying and Working with Service Program References

<http://systeminetwork.com/article/apis-example-identifying-and-working-service-program-references>

**IBM documentation:**

System i ILE Concepts V6R1:

<http://publib.boulder.ibm.com/infocenter/systems/topic/books/sc415606.pdf>

Redbook: Moving to Integrated Language Environment for RPG IV:

<http://www.redbooks.ibm.com/redbooks/pdfs/gg244358.pdf>

**This article demonstrates the following Program and CL Command APIs:**

Retrieve Module Information (QBNRMODI) API:

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

List Module Information (QBNLMODI) API:

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

Retrieve Program Information (QCLRPGM) API:

<http://publib.boulder.ibm.com/infocenter/iserics/v5r4/topic/apis/qclrpgmi.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/57348\\_710\\_WrkPgmImp.zip](http://www.pentontech.com/IBMContent/Documents/article/57348_710_WrkPgmImp.zip)

**Source URL:** <http://iprodeveloper.com/application-development/apis-example-locating-and-working-module-imports>