

DYNAMIC SERVICE PROGRAM BINDING

OVERVIEW

Original Program Model (OPM) Call operations are known as dynamic program call. This means that OS/400 resolves the link between the 2 programs at runtime (not compile time). The called program does not even need to exist when the calling program is compiled. As long as we do not change the parameters (& attributes) used to link the 2 programs, we may recompile either program without affecting each other.

Dynamic program can be considered “modularize”, but to some extent a cost is associated with this. OS/400 resolves all the references at runtime (program addresses, spaces & authorities). Performance suffers if you have many program calls.

To minimize performance degradation, ILE lets us bound procedures within a program. This is known as static call. To call procedures in ILE, the call operation used is:

- CALLB for ILE RPG
- CALLP for ILE RPG. The difference between CALLP & CALLB is that CALLB uses a traditional syntax to call a static, or bound, procedure. CALLP and the function call allow free-format coding of a bound call; in addition, the function call allows a return value from the called procedure.
- CALLPRC for ILE CL
- CALL LINKAGE PROCEDURE for ILE COBOL
- Function call for ILE C

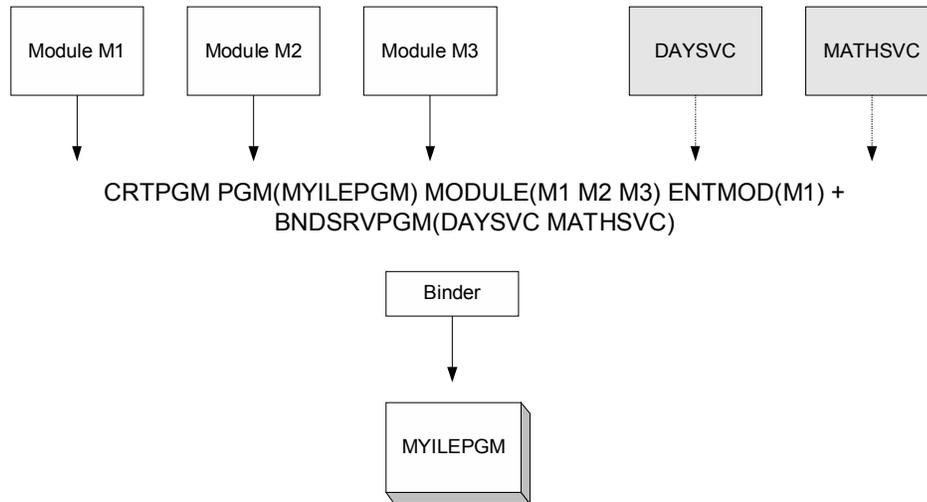
Static call performs better than dynamic call because steps required resolving the references at runtime (in OPM) are done during the binding steps (in ILE).

Binder Function

The binder function is similar to a Linkage function. The binder processes Import request from procedure names. Then it tries to find matching Exports within the modules, service programs and Binding Directory.

When creating an ILE program/service program, 2 types of binding are executed:

- Bind by Copy. To create ILE program/service program, the modules (specified on the create command) are copied. Whenever binding directory is used all unresolved imports are selected within the directory.
- Bind by Reference. Symbolic links to the service programs (that provide exports to unresolved imports) are stored in the created program/service program. The links are physical addresses where the service programs are activated.



The binder uses modules M1, M2, and M3, and service programs DAYSVK and MATHSVK to create the ILE program MYLEPGM.

DYNAMIC BINDING AT RUN-TIME

The traditional way to combine two parts of an application on the AS/400 is to put a piece in one program (*PGM) object, put another pieces in another program; and call back and forth between the main entry points of the programs. This is traditionally known as dynamic program calls. However, calling a program is slower than calling an internal procedure, so many programmers gave up function separation. In V2R3, the AS/400 introduced the service program (*SRVPGM) object, which enables both separation and high performance. But the choice of a service program's name and procedure's name couldn't be decided on runtime.

This type of flexibility has been available through PC executable files called Dynamic Link Libraries (DLL). So to create a similar way to call AS/400 application routines with the flexibility of dynamic program calls and the speed of bound procedure calls, we created this function to do Dynamic Service Program Binding.

When you create a bound program (either a program or a service program), it's bound to a specific list of modules. You can't change this list without recreating the bound program. You can, however, create a program that is bound to service programs in library *LIBL, and this ability gives you some flexibility in deciding which service program is used at runtime. However, the service program found at runtime must have the same list of exports as the service program used to create it. – Use the DSPPGM (Display Program) command with parameter OPTION(*SRVPGM) to display the list of service programs directly bound to a program, and use the DSPSRVPGM (Display Service Program) command with parameter OPTION(*SRVPGM) to display the list of service programs directly bound to a service program.

A service program exports symbols (procedure and variable names) so other bound programs can bind to and use them. Any symbol the service program doesn't export is hidden inside the service program. When you create a service program, you specify which symbols are exported. The CRTSRVPGM (Create Service Program) keyword EXPORT(*ALL) causes all symbols to be exported. To control exactly which symbols are exported, you list the symbols in a QSRVSRC source file member and use keyword EXPORT(*SRCFILE). For ease of maintenance, exporting *ALL symbols is the way to go. But for production, it is sometime best to explicitly export only those symbols you intend other programs to use. You can add or delete hidden procedures or variables in your code without affecting any other program. Adding or deleting an exported symbol may force you to re-create all programs that are bound to it.

This section discusses the technique of doing “Dynamic binding” at run-time (as opposed to compile time). This is not to be confused with Dynamic Call (call to OPM). Why dynamic binding? With dynamic binding, we can delay activation of service programs. For example, we have a service program that contains 100 procedures, but your program uses the procedures based on condition and this changes any time. Without dynamic binding, you have to bind this service program to your main program. Activate the service program every time your program starts. The main program waits until all 100 procedures of the service program have been activated. With dynamic binding, we decide when to activate and what procedure to activate to.

GETSPGMPTR - Get (and Activate) Service Program Pointer

We created the service program GETSPGMPTR. Whenever we need to dynamically activate a service program, we need to compile the main program with GETSPGMPTR service program (see

Setting up your Main Module). GETSPGMPTR uses 3 functions to accomplish dynamic binding. They are – MI rslvsp function, and C QleActBndPgm and QleGetExp functions:

The rslvsp (Resolve System Pointer) function locates the AS/400 object and stores the object’s address using system pointer.

QleActBndPgm (Activate Bound Program) and QleGetExp (Get-Export) are C-functions used by the CRTPGM command. QleActBndPgm API activates the bound program (or service program) specified by the system pointer (retrieved from rslvsp). QleActBndPgm returns an “Activation mark”. QleActBndPgm is normally used in conjunction with QleGetExp API. First you activate the service program, then later, you use QleGetExp to retrieve pointers to procedures and data from the activation. QleGetExp gets the pointer to the “exported” procedure name. QleGetExp requires an “Activation mark” for input (retrieved from QleActBndPgm).

Setting up your Main Program

GETSPGMPTR accepts 3 parameters: the service program name, service program library name and the procedure name within the service program. GETSPGMPTR returns the system pointer of the procedure name. The main program calls the external procedure with the use of this procedure pointer. You would specify the procedure pointer with the keyword EXTPROC. Here is a sample of defining GETSPGMPTR and using F_BUSDATE procedure.

```

D GetSPgmPtr      PR          *    ProcPtr
D   SrvPgm        10A        Const
D   Library       10A        Const
D   Procedure     100A       Const      Varying

D DateBus         PR          10A    ExtProc(#DateBus)
D   Date_Fmt      4A         Value
D   #DateBus      S          *      ProcPtr

D DateReq         S          10A

*
C          Eval    SrvPgm    = 'DATEBUS '
C          Eval    Lib      = '*LIBL '
C          Eval    Procedure = 'F_BUSDATE '
C          Eval    #DateBus  = GetSPgmPtr (SrvPgm:Lib:
C                                     Procedure)
C          Eval    DateReq   = DateBus (*MDY)

```

In the example above, the service program GETSPGMPTR is defined in the data definition. GETSPGMPTR returns a procedure pointer (PROCPTR keyword). The receiver variable #DATEBUS is defined as a System Pointer. The service program I am dynamically binding is *LIBL/DATEBUS. Within the main program, I am trying to activate F_BUSDATE procedure. This procedure return to the main program the business date. The date format I want returned is in *MDY date format.

The step above shows how we dynamically activate the procedure F_BUSDATE. The service program DATEBUS may include more than 1 procedure. Changing the procedure name and executing the GETSPGMPTR function will dynamically activate the new procedure.

It is strongly recommended to execute GETSPGMPTR function once for every procedure you want activated. You would normally initialize a pointer to *NULL. After GETSPGMPTR is executed for the procedure, the procedure pointer will be set to the proper address.

There is no API presently that will de-activate a procedure once activated. Activation gets deleted when their activation group gets deleted (e.g. RCLACTGRP), or whenever the job ends. By default, a service program is created with a keyword of ACTGRP(*CALLER). This value means a service program runs in the same activation group as the program that called it. This activation group is usually the same activation group as the main program. Note: If you want to explicitly de-activate the procedure, the C function "exit()" can be used as an external procedure (ExtProc).