The Business, Science and Uses of ILE Service Programs

Charles Guarino
Central Park Data Systems, Inc.

About The Speaker

With an IT career spanning over 30 years, Charles Guarino has been a consultant for most of them. Since 1995 he has been founder and President of Central Park Data Systems, Inc., a New York area based IBM midrange consulting company. In addition to being a professional speaker, he is a frequent contributor of technical and strategic articles and webcasts for the IT community. He is a proud member of COMMON’s Speaker Excellence Hall of Fame and also Long Island Software and Technology Network’s Twenty Top Techies of 2009. Charles currently serves as a member of COMMON’s Strategic Education Team (SET) and is also Immediate Past President and monthly Q&A host of LISUG, a Long Island IBM i User’s Group www.lisug.org. Charles can be reached at cguarino@centralparkdata.com. LinkedIn - http://www.linkedin.com/in/guarinocharles Twitter - @charlieguarino
In the beginning, your system is stable with no major incidents

- ItemInq
- ItemMaint
- ItemTransfer

Business Disruption !!!
This merger brings new requirements to our programs

1) Unit of measure quantity conversions

2) Proprietary industry-specific conversion and product mixing routines

3) Need to make these conversions available to all programs with the exception of the proprietary conversion routines.

4) Additional requirements to be defined in the future

5) We need to expose these routines down the road to external users via a web service

Technique #1

- Modify mainline code for each program
- ItemInq
- ItemMaint
- ItemTransfer
<table>
<thead>
<tr>
<th>Technique #1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Advantages</strong></td>
<td>This space intentionally left blank</td>
</tr>
<tr>
<td><strong>Disadvantages</strong></td>
<td>Potential conflict with other global variables</td>
</tr>
<tr>
<td></td>
<td>Add’l modifications may not be consistent across programs</td>
</tr>
<tr>
<td></td>
<td>Original modification may be diluted as new functionality is added to each program</td>
</tr>
<tr>
<td></td>
<td>Cannot protect certain subprocedures from being executed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Technique #2 - Modularize</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mod·u·lar·ize</strong> [moj-uh-luh-rahyz]</td>
<td></td>
</tr>
<tr>
<td><strong>verb (used with object), mod·u·lar·ized, mod·u·lar·iz·ing.</strong></td>
<td>to form or organize into modules, as for flexibility.</td>
</tr>
</tbody>
</table>
Possible solution #1

Write individual sub-programs for each function and call from the existing programs.

- ItemInq
- ItemMaint
- ItemTransfer

Units ToCases
Cases ToUnits
Industry Specific
Possible solution #1

Advantages

This space intentionally left blank

Disadvantages

- See disadvantages of possible solution #1
- Still a maintenance nightmare
- Potential performance issues with dynamic binding
- Can’t take advantage of activation group scoping or recursive calls

February 16th, 1993 – A BIG announcement by IBM
Do you remember where you were?

A NEW GENERATION: IBM BOOSTS AS/400 POWER UP TO 60 PERCENT, ENHANCES OPENESS, CLIENT SERVER CAPABILITIES

“...The new release of Operating System/400 Version 2 Release 3 provides enhanced system facilities, ... and a new Integrated Language Environment (ILE) for programming languages.

Also announced was the first ILE programming language, ILE C/400...

In a Statement of Direction, IBM said it intends to provide ILE RPG/400 and ILE/COBOL.

With ILE and enhanced languages, developers can increase their flexibility and productivity by using the most efficient language for a particular module of a program.”
Possible solution #2  - program ITEMINQA

```plaintext
ctl-opt dfmactgrp("no");
// This example has the sub procedures locally defined.
dcl-s item char(20);
dcl-s cases zoned(9:2);
dcl-s units zoned(9:2);  

// Do some very complex code here and then use conversion routines
//============================================================================

dcl-pr CasesToUnits;
  item char(20) const;
  cases zoned(9:2) const;
  units zoned(9:2);
end-pr;

  CasesToUnits (item : cases : units);
  "init" = "nn"
  return;

// =============================================================================
// This procedure converts cases to units quantity
// =============================================================================
dcl-proc CasesToUnits export;
```

Possible solution #2

**Advantages**

Isolation of new program logic

**Disadvantages**

- See disadvantages of possible solution #1
- Still a maintenance nightmare
Possible solution #3 – Create external module

1) Units to cases conversion
2) Cases to units conversion
3) Industry specific algorithms

You will use CRTRPGMOD !!!

Possible solution #3 - Module MASTERCON1

ctl-opt module;

dcl-pr UnitsToCases;
  item    char(20)  const;
  unitsin zoned(9:2)  const;
  casesout zoned(9:2);
end-pr;

dcl-pr CasesToUnits;
  item    char(20)  const;
  casesin zoned(9:2)  const;
  unitsout zoned(9:2);
end-pr;

// ******************************************************************************
// this procedure converts units to case quantity
// ******************************************************************************
dcl-proc UnitsToCases export;
  dcl-pi \n;
  item    char(20)  const;
  units   zoned(9:2)  const;
  casesout zoned(9:2);
end-pi;
Possible solution #3 - DSPMOD MASTERCON1

Use CRTPGM to combine the modules of the existing programs with the new external module.

Module ITEMINQB + Module MASTERCON1

= Program ITEMINQB
Possible solution #3a - Module ITEMINQB

```c
ctl opt;

dcl-s item char(20);
dcl-s cases zoned(9:2);
dcl-s units zoned(9:2);

// **************************************************************************
// Do some very complex code here and then use conversion routines
// **************************************************************************

dcl pr CasesToUnits;
   iim    char(20) iim1;
   cases  zoned(9:2) const;
   units  zoned(9:2);
end-pr;

   CasesToUnits (item : cases: units);

   *inlr = *on;
   return;
```

Possible solution #3a

Create the modules

CRTRPGMOD MODULE(XMLLIB/ITEMINQB) SRCFILE(XMLLIB/QRPGLESRC) DBGVIEW(*SOURCE)

CRTRPGMOD MODULE(XMLLIB/MASTERCON1) SRCFILE(XMLLIB/QRPGLESRC) DBGVIEW(*SOURCE)

Create the program

CRTPGM PGM(XMLLIB/ITEMINQB) MODULE(XMLLIB/ITEMINQB XMLLIB/MASTERCON1)
Possible solution #3a  - DSPPGM ITEMINQB

### Advantages

Statically bound modules do execute more quickly

### Disadvantages

- Newer hardware diminishes the performance argument
- Must bind *ALL modules to create usable programs
- Future modifications will require re-binding of *ALL programs
Possible solution #3b – Bind modules w/ binding directory

A Binding Directory is nothing more than a table of contents for your modules (and service programs).

It provides an easy way to reference these objects.

<table>
<thead>
<tr>
<th>Possible solution #3b</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Create the binding directory</strong></td>
</tr>
<tr>
<td>CRTBNDDIR BNDDIR(XMLLIB/UTILITIES)</td>
</tr>
<tr>
<td><strong>Add the modules as a directory entry</strong></td>
</tr>
<tr>
<td>ADDRNDIRF BNDDIR(XMLLIB/UTILITIES) OBJ((XMLLIB/MASTCON1 *MODUL))</td>
</tr>
<tr>
<td><strong>Reference the binding directory in your program</strong></td>
</tr>
<tr>
<td><code>ctl-opt bnddir('UTILITIES') dftactgrp(*no);</code></td>
</tr>
<tr>
<td><strong>Compile your program, CRTPGM not required</strong></td>
</tr>
<tr>
<td>CRTBNDRPG PGM(XMLLIB/ITEMINQC) SRCFILE(XMLLIB/QRPGLESRC) DBGVIEW(*SOURCE)</td>
</tr>
</tbody>
</table>
Possible solution #3b - Program ITEMINQC

```c
ctl-opt bnddir('UTILITIES') !fctctgrp('no');

dcl-s item char(20);
dcl-s cases zoned(9:2);
dcl s units zoned(9:2);

// ************************************************************************
// Do some very complex code here and then use conversion routines
// ************************************************************************

dcl-pr CasesToUnits;
   item char(20) const;
   cases zoned(9:2) const;
   units zoned(9:2);
end-pr;

   CasesToUnits (item : cases : units);

   *inlr = "on";
   return;
```

Possible solution #3b - WRKBNDIRED UTILITIES

```
Work with Binding Directory Entries

Binding Directory: UTILITIES  Library: XMLLIB

Type options, press Enter:
1=Add  4=Remove

--------Creation--------

<table>
<thead>
<tr>
<th>Opt</th>
<th>Object</th>
<th>Type</th>
<th>Library</th>
<th>Activation</th>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MASTERCONI</td>
<td>MODULE</td>
<td>XMLLIB</td>
<td></td>
<td>03/02/14</td>
<td>18:24:44</td>
</tr>
</tbody>
</table>
```
Possible solution #3b - DSPPGM ITEMINQC

Advantages

- Statically bound modules do execute more quickly
- The binding directory makes it easier to compile programs

Disadvantages

- Newer hardware diminishes the performance argument
- Must compile *ALL programs to include usable modules
- Future modifications will require recompiling of *ALL programs
In these scenarios, a separate copy of module MASTERCON1 is included in every program.

Business Disruption !!!
Now you need to RE-BIND *ALL !!!
Either Re-compile or use the UPDPGM command for EVERY pgm

Enter the Service Program

- Created from a module
- Cannot be directly executed
- Has an object type of *SRVPGM
- Used in conjunction with binder language
- Uses a **signature** to validate callers
- Don’t need to rebind all programs that use it when a change is made
- Are statically bound by reference, not statically bound by copy
- Can contain one or more modules
- I definitely prefer them over using *modules
The simple* start of a service program

- CRTRPGMOD
- CRTRPGMOD using EXPORT *ALL
- A current signature is automatically assigned
- Add the service program to a binding directory
- Compile the program that references the service program (the compiled program will remember the current signature)
- Call your program and the service program is activated. (if a valid signature is not found you will have a violation)

  * Simple but not always the best

The listing detail option…. 

![Image of listing detail option]
The Steps...

CRTRPGMOD MODULE(XMLLIB/MASTERCONV) SRCFILE(XMLLIB/QRPGLESRC) SRCMBR(MASTERCONV) DBGVIEW(*SOURCE) REPLACE(*YES)

CRTRPGMOD SRCPGM(XMLLIB/MASTERCONV) EXPORT(*ALL)

DSPSRPGM MASTERCONV

Procedure Exports:

<table>
<thead>
<tr>
<th>Procedure Name</th>
<th>ARGCPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASESTOUNITS</td>
<td>*NO</td>
</tr>
<tr>
<td>UNITSTOCARCRC</td>
<td>*NO</td>
</tr>
</tbody>
</table>

Detail: *SIGNATURE

Signatures:

G0960086E2E017F1F1644210E41F5F114

The Steps...

ADDBNDDIRE BNDDIR(XMLLIB/UTILITIES) OBJ((XMLLIB/MASTERCONV))

WRKBNDDIRE UTILITIES

Binding Directory: UTILITIES Library: XMLLIB

Type options press enter

L=Add *R=Remove

Date Time

- MASTERCONV SRVPGM XMLLIB *IMMED 02/20/14 22:30:39
The Steps... Program ITEMINQD

```
ctl-opt bnddir('UTILITIES') dftactgrp("no");

dcl-i item char(20);
dcl-e cases zoned(9:2);
dcl-s units zoned(9:2);

// ******************************************************************************
// Do some very complex code here and then use conversion routines
// ******************************************************************************

dcl-pr CasesToUnits;
  item char(20) const;
  cases zoned(9:2) const;
  units zoned(9:2);
end-pr;

CasesToUnits (item : cases : units);

*inir = "on";
return;
```

The Steps...

```
CRTBNDRPG PGM(XMILLIB/ITEMINQD) SRCFILE(XMILLIB/QRPGLESRC) SRCMBR(ITEMINQD)
DBGVIEW("SOURCE") REPLACE("YES")

DSPGPM ITEMINQD
```

<table>
<thead>
<tr>
<th>Opt</th>
<th>Service</th>
<th>Program</th>
<th>Library</th>
<th>Activation</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>NMS/SPRDCNV</td>
<td>*LWNL</td>
<td>*IMMLED</td>
<td>080881328730515154143151451415315</td>
<td>44F79F3B98069533678DF0CF105F8EC1</td>
</tr>
<tr>
<td>-</td>
<td>QNWXIE</td>
<td>QSYS</td>
<td>*IMMED</td>
<td>58BD5573C8C5904040380443BE0D5</td>
<td>44F79F3B98069533678DF0CF105F8EC1</td>
</tr>
<tr>
<td>-</td>
<td>QLEAHK</td>
<td>QSYS</td>
<td>*IMMED</td>
<td>58BD5573C8C5904040380443BE0D5</td>
<td>44F79F3B98069533678DF0CF105F8EC1</td>
</tr>
</tbody>
</table>
About that signature…

A.K.A. 0000000E2E017F1F1044210E41F5F114

In these scenarios, as long as the signature matches when the service program is called, everything will be fine.
Business Disruption !!!

Added two more procedures to MASTERCONV

```plaintext
ctl-opt nameline;

dcl-pr UnitsToCases;
  item   char(20)   const;
  unitsin zoned(9:2) const;
  casesout zoned(9:2);
end-pr;


dcl-pr CasesToUnits;
  item   char(20)   const;
  casesin zoned(9:2) const;
  unitssout zoned(9:2);
end-pr;


dcl-pr UnitsToPallets;
  item   char(20)   const;
  unitsin zoned(9:2) const;
  palletssout zoned(9:2);
end-pr;


dcl-pr PalletsToUnits;
  item   char(20)   const;
  palletsin zoned(9:2) const;
  unitssout zoned(9:2);
end-pr;
```
What happens to the signature?

CRTSRVPGM SRVPGM(XMLLIB/MASTERCONV) EXPORT(*ALL)

Ludwig van Beethoven

William Zhurbysce

DSPSRVPGM MASTERCONV

Procedure Exports:

<table>
<thead>
<tr>
<th>Procedure Line</th>
<th>ACCEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASESTOUNITS</td>
<td>*NC</td>
</tr>
<tr>
<td>PALLETS2SPANS</td>
<td>*NO</td>
</tr>
<tr>
<td>UNITSPACKAGES</td>
<td>*NO</td>
</tr>
<tr>
<td>UNITSPALLETS</td>
<td>*NO</td>
</tr>
</tbody>
</table>

EXPORT(*ALL) exports in alphabetical order

William Zhurbysce
What happens now that the three programs who know “Beethoven” call a “Shakespeare” service program?

How do you fix this?

<table>
<thead>
<tr>
<th>Selection or command</th>
<th>==&gt; call iteming</th>
</tr>
</thead>
<tbody>
<tr>
<td>F3=Exit  F4=Prompt  F9=Retrieve  F12=Cancel  F13=</td>
<td></td>
</tr>
<tr>
<td>F23=Set Initial menu</td>
<td></td>
</tr>
<tr>
<td>Error found on CALL command.</td>
<td></td>
</tr>
</tbody>
</table>

3> call iteming

Program signature violation.
Error found on CALL command.

| Message ID . . . . . . . : MLH4431  Severity . . . . . . : 40 |
| Message type . . . . . : Escape  Data sent . . . . . . : 02/01/14  Time sent . . . . . . : 10:37:30 |

Message . . . : Program signature violation.  Cause . . . . . : The source program ITEMING specifies a signature X'000000002E017F1F154E010EE41FF114', which is not supported by service program MASTERCONV.  Recovery . . . : The service program interface has changed. Re-bind source program ITEMING.
Possible signature violation solution # 1

UPDPGM PGM(XMLLIB/ITEMINQ) MODULE(*NONE)

UPDPGM PGM(XMLLIB/ITEMMAINT) MODULE(*NONE)

UPDPGM PGM(XMLLIB/ITFMXFFR) MODULE(*NONE)

This solution works if you can identify EVERY program referencing the service program.

Also, if done manually will be a VERY tedious process.

Possible signature violation solution # 2

Retrieve the binder source BEFORE you make any changes !!!

RTVBNDSRC SRVPGM(XMLLIB/MASTERCONV) SRCFILE(XMLLIB/QSRVSRC)

STRPGMEXP PGMLVL(*CURRENT)
EXPORT SYMBOL('CASESTOUNITS')
EXPORT SYMBOL('UNITSTOCASES')
ENDPGMEXP
About that signature…

`CRTSRVPGM SRVPGM(XMLLIB/MASTERCONV) EXPORT(*ALL)`

Will produce the same current signature as

`CRTSRVPGM SRVPGM(XMLLIB/MASTERCONV) SRCFILE(XMLLIB/QSRVSRVRC)`

```plaintext
STRPGMEXP PGMLVL(*CURRENT)
    EXPORT SYMBOL('CASESTOUNITS')
    EXPORT SYMBOL('UNITSTOCASES')
ENDPGMEXP
```

"0000000E2E017F1F1044210E41F5F114"

For your existing bound programs to continue working the signature still needs to match

```plaintext
STRPGMEXP PGMLVL(*CURRENT)
    EXPORT SYMBOL('CasesToUnits')
    EXPORT SYMBOL('UnitsToCases')
    EXPORT SYMBOL('UnitsToPallets')
    EXPORT SYMBOL('PalletsToUnits')
ENDPGMEXP

STRPGMEXP PGMLVL(*PRV)
    EXPORT SYMBOL('CASESTOUNITS')
    EXPORT SYMBOL('UNITSTOCASES')
ENDPGMEXP
```

"0000000E2E017F1F1044210E41F5F114" will now become the *PRV

A NEW signature will be generated with the *CURRENT export
Now service program MASTERCONV has TWO signatures!

Now MASTERCONV has two signatures

*CURRENT ...2317

*PRV ...F114
May I see your ID?

Different *CURRENT signatures, SAME *PRV signatures

```
<table>
<thead>
<tr>
<th>STRPGMEXP</th>
<th>PGMLVL(*CURRENT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPORT</td>
<td>SYMBOL(CasesToUnits)</td>
</tr>
<tr>
<td>EXPORT</td>
<td>SYMBOL(UnitsToCases)</td>
</tr>
<tr>
<td>EXPORT</td>
<td>SYMBOL(UnitsToPallets)</td>
</tr>
<tr>
<td>EXPORT</td>
<td>SYMBOL(PalletsToUnits)</td>
</tr>
<tr>
<td>ENDPGMEXP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STRPGMEXP</th>
<th>PGMLVL(*CURRENT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPORT</td>
<td>SYMBOL(CasesToUnits)</td>
</tr>
<tr>
<td>EXPORT</td>
<td>SYMBOL(UnitsToPallets)</td>
</tr>
<tr>
<td>EXPORT</td>
<td>SYMBOL(PalletsToUnits)</td>
</tr>
<tr>
<td>ENDPGMEXP</td>
<td></td>
</tr>
</tbody>
</table>

G. Andres

Ludwig Van Beethoven

J. I. Kennedy

Ludwig Van Beethoven
```
STRPGMEXP prompted

Using named signatures

```
STRPGMEXP PGMLVL(*CURRENT) SIGNATURE('MOZART')
EXPORT SYMBOL(CasesToUnits)
EXPORT SYMBOL(UnitsToCases)
EXPORT SYMBOL(UnitsToPallets)
EXPORT SYMBOL(PalletsToUnits)
ENDPGMEXP

STRPGMEXP PGMLVL(*PRV) SIGNATURE('BEETHOVEN')
EXPORT SYMBOL('CASESTOUNITS')
EXPORT SYMBOL('UNITSTOCASES')
ENDPGMEXP
```
Using named signatures

CRTRSVPGM SRVPGM(XMLLIB/MASTERCONV) SRCFILE(XMLLIB/QSRVSRV)

DSPSRVPGM MASTERCONV DETAIL(*SIGNATURE)

Signatures:
D4B0E6C1DE3494D946D46494D94D40
C2D5C5E3C3883504364844946494D0

Only one signature?

STRPGMFXP PGMIVI (*CURRENT) SIGNATURF(MASTERSTG)
EXPORT SYMBOL(CASESTOUNITS)
EXPORT SYMBOL(UNITSTOCASES)
EXPORT SYMBOL(UNITSTOPALLETS)
EXPORT SYMBOL(PALLETSTOUNITS)
EXPORT SYMBOL(CASESTOPALLETS)
EXPORT SYMBOL(PALLETSTOCASES)

/* Next two added when company ABC was purchased */
EXPORT SYMBOL(UNITSTOINNERPACKS)
EXPORT SYMBOL(INNERPACKSTOUNITS)

/* Next three added when company XYZ was purchased */
EXPORT SYMBOL(UNITSTOCONTAINERS)
EXPORT SYMBOL(CONTAINERTOUNITS)
EXPORT SYMBOL(PALLETSTOCONTAINERS)

ENDPGMEXP
Only one signature?

<table>
<thead>
<tr>
<th>Activation group attribute</th>
<th>*CALLER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared activation group</td>
<td>*NO</td>
</tr>
<tr>
<td>Current export signature</td>
<td>MASTERSIG</td>
</tr>
<tr>
<td>User profile</td>
<td>*USER</td>
</tr>
<tr>
<td>Use adopted authority</td>
<td>*YES</td>
</tr>
</tbody>
</table>

Procedure Name | SRGOPT
---|---
CARGO POINTS | *NO
UNITSTOQCAES | *NO
UNITSTOPALETTS | *NO
PALLESTOPUNITS | *NO
CASESTOPALETTS | *NO
PALLESTOPCAGES | *NO
UNITSTOPINPACK | *NO
IN PACEPOUNITS | *NO
UNITSTOPCONTAINER | *NO
CONTAINER UNITS | *NO

Same sequence as binder source, not alphabetical order

When using multiple service programs

![Image of CRTPGM window]
Wrap up

- Modularization is not a new concept – embrace it!
- EXPORT(*ALL) is OK to use when you are willing to use UPDPGM or re-compile after a service program change.
- There is no hard and fast rule about the number of binding directories or modules entries within them.
- A service program signature can be thought of as a record format level identifier.
- There is a LVLCHK(*NO) parameter on the STRPGMEXP command which acts EXACTLY how you think it would.
- A service program can have more than one signature – BUT – the current one is used as the index.

The Business, Science and Uses of ILE Service Programs

Thank you!!!

Charles Guarino
Central Park Data Systems, Inc.