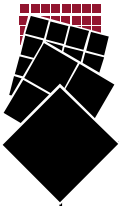


Application modernization without complexity
How to overcome complexity for System i business applications

PART2 ILE or cutting into components

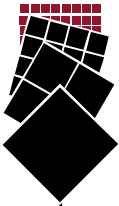
Presented by:
Jean Mikhaleff

Date:
February, 7 2007



General Presentation Part 2

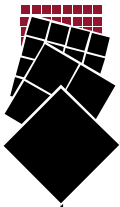
- **Component by function**
- **ILE with an example.**



Into components Part 2

What you have to know to use ILE:

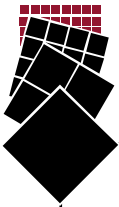
- The principle of an ILE language like a few RPGIV or CLLE.
- To know what a component means.



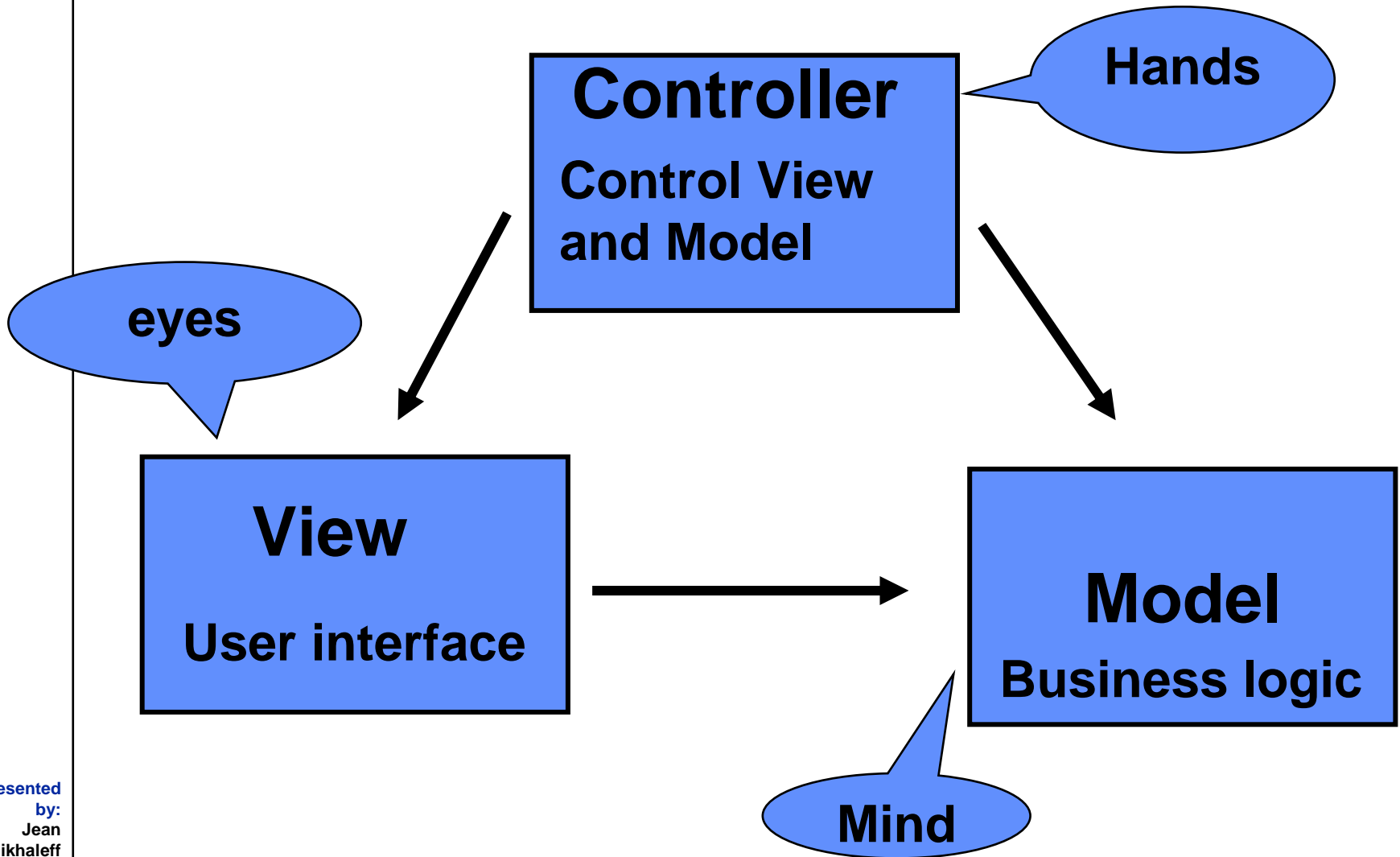
Into components Part 2

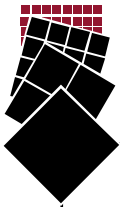
Two Kinds of components

- **By nature or type of instructions like MVC model**
- **By function like object design.**



MVC component Part 2



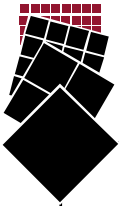


By Function Part 2

We will not see MVC architecture, but ILE by default and ILE objects by function.

Example « customer file » :

- in MVC you have one component to display screen, one component to manage database and another to process and respond user's events.**
- in ILE by default you can have for example a program to list a file with SFL and another program to manage a file with a Form, so you have the same number of programs regarding the number of components.**

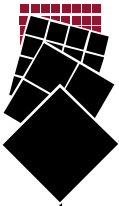


By Function Part 2

Difference between an ILE object *pgm and an ILE by default :

- **An ILE by default program has only one component.**
- **An ILE object *pgm has several component.**

An object is an envelope.



By Function Part 2

Difference between an ILE object *pgm and an ILE by default :

- **An ILE default *pgm has only one component example : one for the customer SFL, one for the customer form... etc....**
- **An ILE object *pgm has several components inside example 10 sfl, one form, etc... To manage completely the « customer data base ». In that case, the *pgm is only a receipt, so the *pgm is based object.**



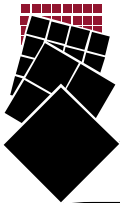
By Function Part 2

Take a very simple example for studying that:

RPGIV: one SFL *pgm and one form *pgm

- These two programs are running very well for years.
- These two programs were compiled with option 14 of PDM : IBM command CRTBNDRPG

```
Program . . . . . > MKSFM1LST
  Library . . . . . > TSTLIBSRC
Source file . . . . . > QRPGLSRC
  Library . . . . . > TSTLIBSRC
Source member . . . . . > MKSFM1LST
```



By Function Part 2

Sfl *pgm
MKSFM1LST

```
unique key                23/01/07  ARCAD02
-----
Client Unique key
-----
C      Client Key . . . . . : 10000
Opt K  Company Name . . . . . : Bank-A-Count Co
2      Adress Line 1. . . . . : 1666 Main St.
1      Adress Line 2. . . . . :
1      City . . . . . : Rudolph
1      State Key. . . . . : WI
1      Zip Postal code. . . . . : 54475
1      Country Code . . . F4: USA  United State of America
1      Phone Number . . . . . : +1 715 435-4616
1      Fax Number . . . . . :
1
1      ENT=Page+1          F12=Previous
1
1
10022  Big Dog Motorcycles, LLC  United State of America
More...
F3=Exit  F5=Refresh  F6=Add  F12=Previous
```

Form *pgm
MKSFM1FCH



By Function Part 2

For each program you have with the **DSPPGM MKSFM1LST** command

One temporary *module in QTEMP

Activation group by default

```
Display Program Information

Program . . . . . MKSFM1LST      Library . . . . .
Owner . . . . . QPGMR
Program attribute . . . . . GLE
Detail . . . . . :          TIC

Program creation information:
Program creation date/time . . . . . : 01/11/06
Type of program . . . . . : ILE
Program entry procedure module . . . . . : MKSFM1LST
Library . . . . . : QTEMP
Activation group attribute . . . . . : *DFTACTGRP
```



By Function Part 2

ENTER and ENTER on DSPPGM MK1SFMLST and you will have one line for the *module

```
Display Progr
Program . . . . . : MKSFMLST
Owner . . . . . : QPGMR
Program attribute . . : RPGLE
Detail . . . . . : *MODULE

Type options, press Enter.
 5=Display description Print de

Opt  Module      Library      Attribute      Creation      Optimization      Debug
   Date          Level
█  MKSFMLST     QTEMP       RPGLE          01/11/06      *NONE              *YES
```

QTEMP *module

Type 5 to see details



By Function Part 2

After 5, you will see the *module attributes

And at the level QTEMP *module details you will find the source file and source library!!!

Module attributes:

```
Module . . . . . : MKSFMLST
  Library . . . . . : QTEMP
Source file . . . . . : QRPGLSRC
  Library . . . . . : TSTLIBSRC
Source member . . . . . : MKSFMLST
Module attribute . . . . . : RPGLE
```



By Function Part 2

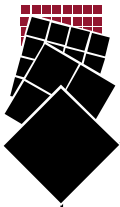
That means :

With the command CRTBNDRPG option 14 of PDM that creates automatically a *pgm,

YOU ARE ILE BUT YOU DON'T SEE IT

Because the *module is in QTEMP AND you have only one temporary *module for one *pgm that share the same name...

AND you have the activation group by default.



By Function Part 2

So, how can I make only one object *pgm with two *module inside?

First step : replace the RPGIV code CALL into CALLB (call boundary)

Second step : create 2 *module in your current library (option 15 of PDM).

Third step: wrap them both in an *pgm envelope and you will have a real object with few changes regarding the RPGIV code!!!



First step CALLB Part 2

Just replace CALL
*pgm into CALLB
*module

SFL program

```
Columns . . . : 6 90 Edit
SEU=> _____
FMT * * . 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ..
606.00 C*
607.00 C          CALLB      'MKSFM1FCH ' WPFch
608.00 C*
609.00 C          IF          WFKeYpress = F03
610.00 C          EVAL        SavKeyP = F
611.00 C          ELSE
612.00 C          EVAL        SavKeyP = ENTER
613.00 C          ENDIF
```

Form program



Second step *module Part 2

Compile the two source members in your current library with option 15 of PDM :

Create *module CRTRPGMOD

Create RPG Module (CRTRPGMOD)

Type choices, press Enter.

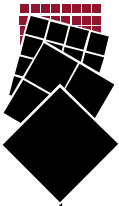
```
Module . . . . . > MKSFM1FCH      Name,  
  Library . . . . . >   TSTLIBOBJ   Name,  
Source file . . . . . > QRPGLESRC   Name,  
  Library . . . . . >   TSTLIBSRC   Name,  
Source member . . . . . > MKSFM1FCH   Name,  
Source stream file . . . . . _____
```



Second step *module Part 2

With the command
DSPOBJD OBJ(MK*) OBJTYPE(*MODULE)

```
Display Object D  
  
Library . . . . . : TSTLIBOBJ  
  
Type options, press Enter.  
  5=Display full attributes  8=Display  
  
Opt  Object          Type          Attribute  
_    MKSFM1FCH       *MODULE      RPGLE  
_    MKSFM1LST       *MODULE      RPGLE
```



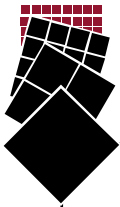
Second step *module Part 2

Question:

- What can I do with these two *module?

Response:

- Nothing at all! Just wrap them together and you'll got a *pgm.



Third step making object Part 2

CRTPGM command

```
. . . PGM > MKSEFM1LST
. . . > TSTLIBOBJ
. . . MODULE > MKSEFM1LST
. . . > TSTLIBOBJ
+ for more values > MKSEFM1FCH
> TSTLIBOBJ
. . . TEXT *ENTMO

Additional Parameters

. . . ACTGRP > COMBELGIUM
```

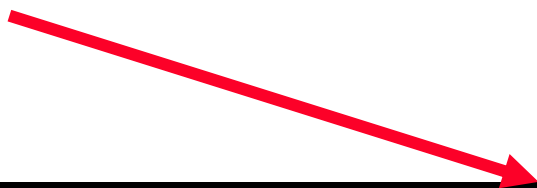
Our two
modules.
SFL is first
called

Don't forget to enter
the activation group
name you want!!!



Third step making object Part 2

DSPPGM + ENTER + ENTER + ENTER



```
Program . . . . . : MKSFM1LST
Owner . . . . . : QPGMR
Program attribute . . : RPGLE
Detail . . . . . : *MODULE
```

Type options, press Enter
5=Display description

Now we have two components in the same envelope. We definitely are object!!!

Opt	Module	Library	Attribute
█	MKSFM1LST	TSTLIBOBJ	RPGLE
—	MKSFM1FCH	TSTLIBOBJ	RPGLE



Third step making object Part 2

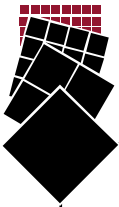
***module SFL**

***module Form**

**Don't forget : CRTPGM
copy the code of the
two *module inside the
*pgm envelope like a
CPYF.**

***module SFL**

***module Form**



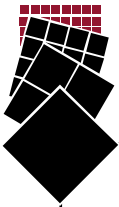
Third step making object Part 2

The aim of object is to cut a big program into functional components in order to maintain easily and to reuse them.

They would both process WITH THE SAME RESPONSE TIME.

Do we have meet this objective with ILE architecture?

Not yet... let's go ahead.



Third step making object Part 2

We created the program with CRTPGM command and the parameter ACTGRP(COMBELGIUM)

When the program MKSFM1LST starts running, it creates a space named COMBELGIUM in which the i5/OS put :

- **all the addresses of DSPF and *module dynamically.**

So after the second call of the MKSFM1FCH component form, the response time will be identical of an EXSR instruction (call internal subroutine).



Third step making object Part 2

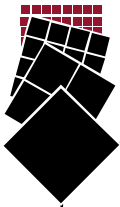
BUT:

- **The main database file CLIENT is open twice! One with the SFL component and one with the form component.**

That costs a lot of time to OPEN and CLOSE a file each time a component is called! One big program still win in response time.

How can we get ride of that?

By creating an other main CLLE component that OPEN and CLOSE the file INSIDE our activation group : COMBELGIUM !!!



Third step making object Part 2

1°) Create *module

```
CRTCLMOD MODULE(TSTLIBOBJ/MKCLIENT)  
SRCFILE(TSTLIBSRC/QRPGLESRC)
```

2°) Create *pgm MKCLIENT

```
CRTPGM PGM(MKCLIENT)  
MODULE(MKCLIENT MKSFM1LST  
MKSFM1FCH)  
ACTGRP(COMBELGIUM)
```

3 *module
and
activation
group



Third step making object Part 2

Display Program Information

```
Program . . . . . : MKCLIENT      Library .  
Owner . . . . . : QPGMR  
Program attribute . . : CLLE  
Detail . . . . . : *MODULE
```

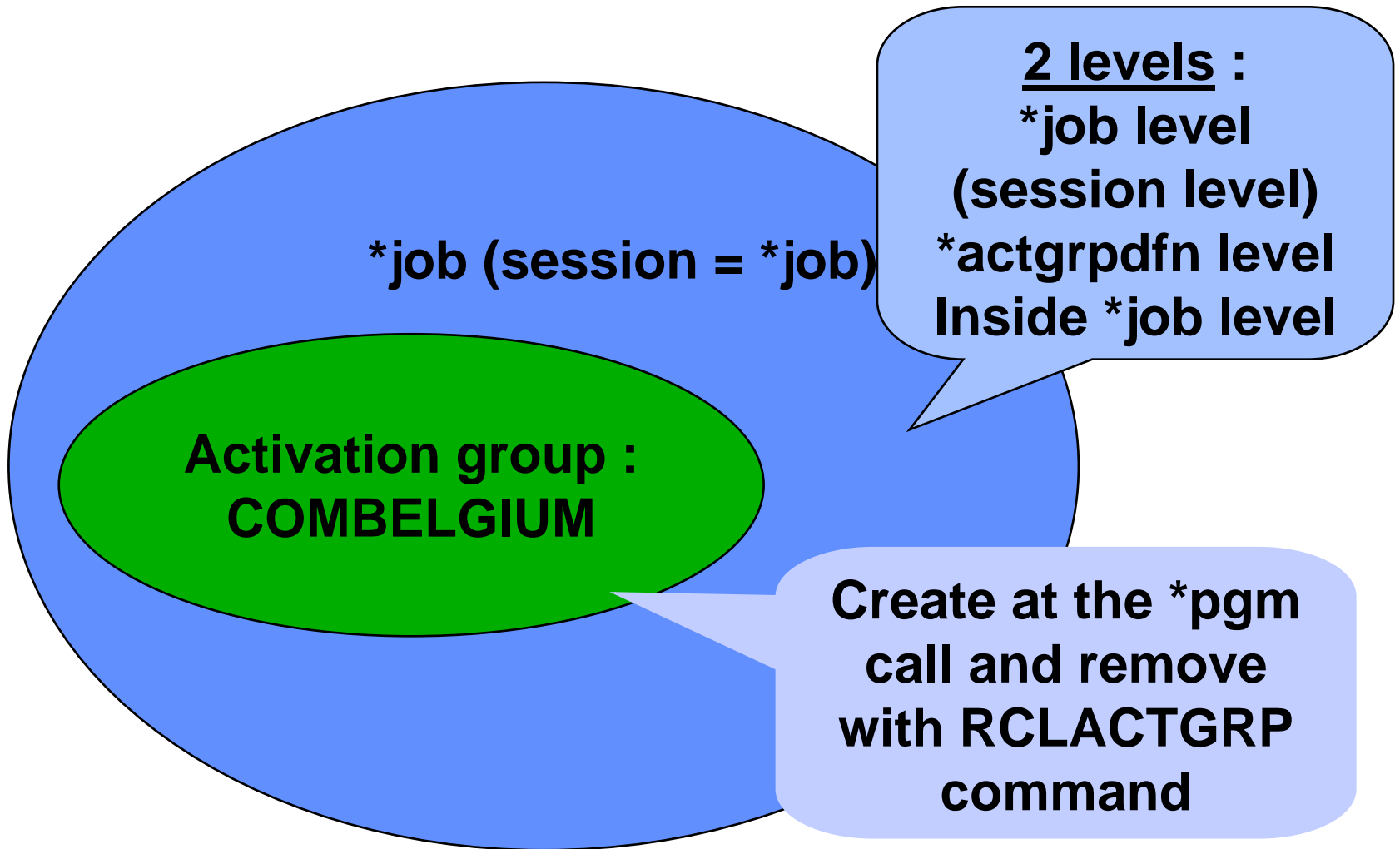
Type options, press Enter.
5=Display description 6

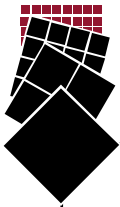
**DSPPPGM+ENTER...
3 *module 1 CLLE and
2 RPGLE**

Opt	Module	Library	Attribute	Creation Date
█	MKCLIENT	TSTLIBOBJ	CLLE	24/01/07
_	MKSFMLST	TSTLIBOBJ	RPGLE	23/01/07
_	MKSFMLFCH	TSTLIBOBJ	RPGLE	23/01/07



Third step making object Part 2





Third step making object Part 2

Our ILE object MKCLIENT is absolutely wrapped into an ILE i5/OS dynamic space called COMBELGIUM with:

- **All the adresses managed inside COMBELGIUM (response time = EXSR after first CALL)**
- **One OPEN/CLOSE for all the components**
- **The components might be an association of CLLE, RPGLE, CBLLE (cobol), SQLRPGLE, SQLCBLLE...**



Third step making object Part 2

All is ok for the best then?

No, not yet... reuse is missing now.

Remember: when a *pgm is created, all the *module are copied like a copyf from a file to another file.

That mean:

- A *module is called structure component because it must not be used more that once in only one program.
- **RULE:** If another program will need the same *module, you have to transform it in order to reuse it.



Third step making object Part 2

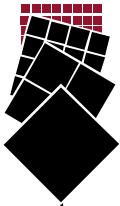
Object *srvpgm:

- is a component like a *module
- has an envelope too, like a *pgm, but cannot directly be called.

So:

- when a *srvpgm is updated, that for *all the *pgm that use it.

A *srvpgm must be created like a *pgm with the instruction CRTSRVPGM. A *srvpgm may have a lot of *module and a lot of *srvpgm too... like a *pgm.



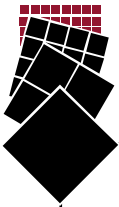
Third step making object Part 2

Example : imagine you want to reuse the form
*module with another *pgm SFL:

1°) you have to transform the *module MKSFM1FCH
into *srvpgm MKSFM1FCH.

2°) you have to re-create the object *pgm
MKCLIENT with the *srvpgm instead of the
*module.

NOTE: you don't need to change the RPGIV at all !!!

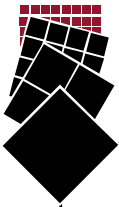


Third step making object Part 2

1°) Create *srvpgm for the form:

```
CRTSRVPGM SRVPGM(TSTLIBOBJ/MKSFM1FCH)  
MODULE(TSTLIBOBJ/MKSFM1FCH)  
EXPORT(*ALL) ACTGRP(*CALLER)
```

The *srvpgm will use the same activation group as its object *pgm



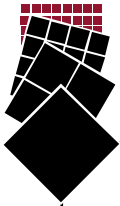
Third step making object Part 2

2°) re-create *pgm MKCLIENT:

```
CRTPGM PGM(TSTLIBOBJ/MKCLIENT)
MODULE(TSTLIBOBJ/MKCLIENT MKSFM1LST)
BNDSRVPGM(MKSFM1FCH)
ACTGRP(COMBELGIUM)
```

Don't forget
the activation
group

*srvpgm, so the form
may be reuse in
another ILE *pgm



General considerations object Part2

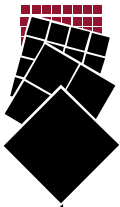
How to manage recursiveness?

Example : SFLA call=> SFLB call=> SFLA

The second level call must failed because SFLA actually call itself. But with activation group it is easy to manage recursiveness!

CRTPGM named SFLA, with parameter :

ACTGRP(*NEW), so new space will be created each time SFLA is called... that mean a new adress each time, like if it was a different *pgm name. And the SFLA *pgm will not failed when calling itself at different levels!



General considerations object Part2

NOTE about OVRDBF and other IBM commands:

Hierarchy of OVRSCOPE parameter :

***JOB until end of job (session is a job)**

***ACTGRPDFN until end of activation group**

***CALLLVL until end of calling component**

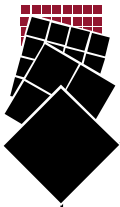
Default value is *ACTGRPDFN... even very few people will use it.



General considerations object Part2

**RULE 1 : USE ILE OBJECTS ONLY TO
MANAGE COMPLEXITY.**

**That means: default *pgm is good enough for
a majority of cases. (don't use it for the
example SFL+Form, that don't worth it at all)**



General considerations object Part2

**RULE 2 : MAKE A BINDING CLP PROGRAM TO
MANAGE CRTSRVPGM AND CRTPGM for easy
lifetime maintainability.**

**(CRTPGM is not a compilation and takes a few
seconds if you already have components online.)**

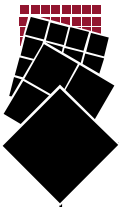


General considerations object Part2

```
CRTSRVPGM SRVPGM(DBWCALL) MODULE(DBWCALL DBWM01 DBWM10 +
  DBWM03 DBWS02 DBWS04 DBWS03 DBWW01 +
  FOTDBW01 DBWM08 INTM08 INTM09 DBWW02 +
  BOXM05 DBWL03 DBWL04 DBWBOXP ADEDBW01 +
  DBLFILE01 DBLFILE02 DBLFILE11 DBLFILE12 +
  DBW_B01_2B ADEDBW11 DBW_B01_1 DBW_B01_3 +
  DBW_S01_1 DBW_S01_2 DBW_S01_3 DBW_S01_4 +
  DBWM20 DBWM19 DBW_S01_0 DBWM12 DBWM13) +
  EXPORT(*ALL) BNDSRVPGM(WSTM07 WSTM15 +
  SYSVAL03 DBWCPY01 DBWCPY02 DBWCPY03 +
  PULDBW01 PULDBW02 INTM04 DBWM04 PDMOF_01 +
  LSTM03 WRQCLSCAN DBWL05 ATO01 PDMOF_02 +
  DBW_R01_1 DBW_B01_4 DBW_B01_2 DBLDBL02 +
  DBLDSPF01 DBLDSPF02 DBLFILD11 DBLFILE51 +
  DBWM18 DBLFILE52 DBWM05 DBWM11 DBWM14 +
  LSTF08D DBWM16 DBWM17) ACTGRP(DBW)
```

**This
*srvpgm
has 35
*module
and 31
*srvpgm.**

**When one component is changed, the
CRTSRVPGM is done by calling a binding CLP**



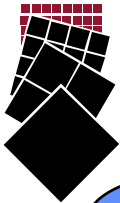
General considerations object Part2

RULE 3 : Test bottom top.

when updating a component, create a temporary default *pgm (option 14 of pdm) in order to test it.

Then do the binding and test the high level component.

Then test the application.



General considerations object Part2

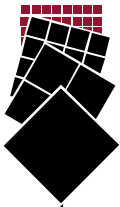
Motor: high level reuse component like *srvpgm

A truck is an object that works by itself like *pgm



The wheel is a reuse component like *srvpgm

Chassis is a structure component like *module



General considerations object Part2

The only one method to deal with complexity is to divide into functional components, like the objects and the components of the industry world. (i.e. Intel inside)

If this method works well for the industry objects that will works for softwar objects too...

With ILE, based object is i5/OS integrated.

Enjoy using it!

Jean Mikhaleff thanks for your attention