



Modern CL Programming



OMNI Users group
January 21, 2020.



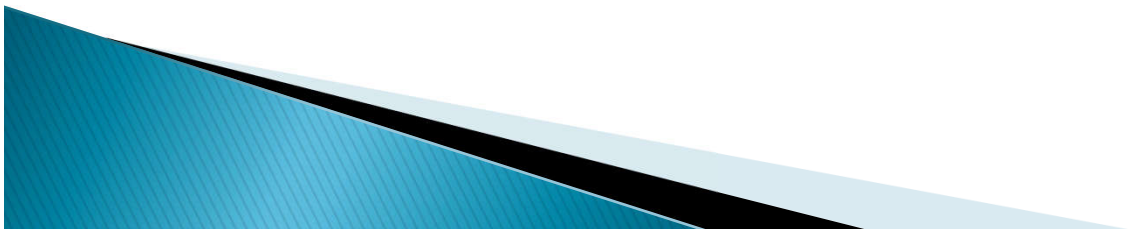
Larry "DrFranken" Bolhuis
Frankeni Technology Consulting, LLC
Grand Rapids, MI

lbolhuis@frankeni.com
616.855.1667
www.frankeni.com

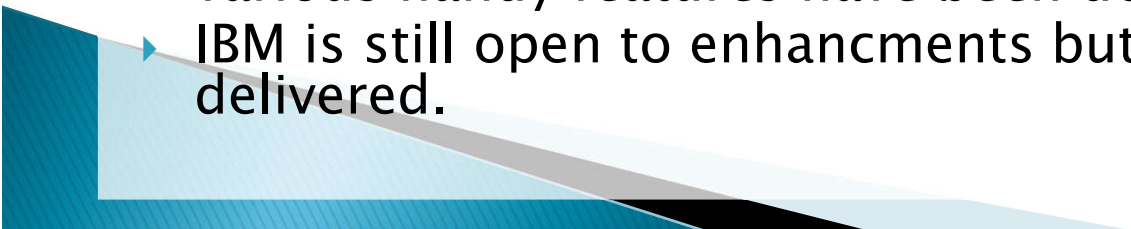


Why are we here?

- ▶ This ain't your father's CL!
- ▶ CL is capable of being written in a way that is much more modular than ever before.
 - Use of the dreaded "goto" is relegated to history!
- ▶ Standard constructs that you've probably used in other languages for years are now properly part of CL.
- ▶ We will work our way through the parts of a CL program and look at the things CL didn't use to have that enable more modern code to be used.
- ▶ *CMD (Command) objects will be covered as well.

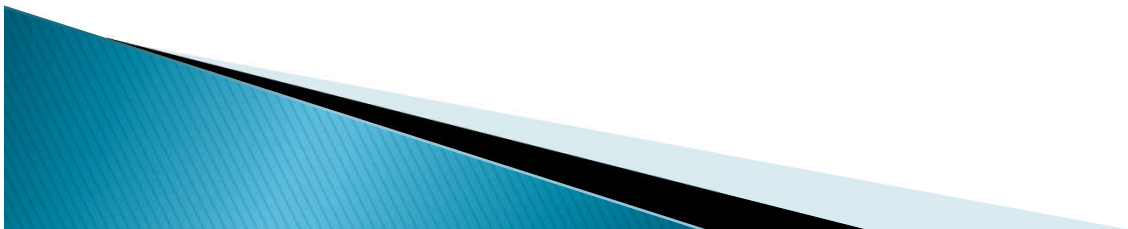


A Brief history of recent CL Enhancements:

- ▶ In OS/400 V5R1 GUI command prompting was added to iNav and other interfaces. This was 'cool.'
 - Implementation is XML to a Java applet. Used in WDSC, iNav, Access for web, and others.
 - ▶ In OS/400 V5R2 the ability to digitally sign your command objects was introduced. This was 'a waste of time.'
 - It was the only thing CL got in V5R2... ☹
 - ▶ In i5/OS V5R3 we got new data types, increased parameter lengths and counts, new commands and more. These are 'Awesome'
 - ▶ In IBM i 5.4 a continuation of what was delivered in V5R3 is provided. This is 'Spectacular.'
 - ▶ In IBM i 6.1 previous enhancements have been enhanced! This is 'Encouraging'.
 - ▶ In IBM i 7.1 more enhancements arrive. This was 'Amazing.'
 - ▶ Since then the major language enhancements have ended but various handy features have been delivered.
 - ▶ IBM is still open to enhancements but the big items are delivered.
- 

Agenda:

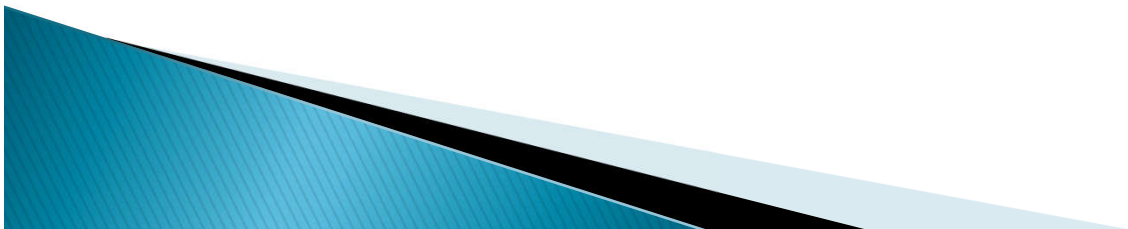
- ▶ Variable Types
- ▶ Parameter enhancements
- ▶ Multiple File Support
- ▶ Declare Processing Options
- ▶ Source member Include
- ▶ Control Flow Enhancements
- ▶ Subroutines
- ▶ Command Enhancements
- ▶ New API QCAVFYNM
- ▶ Proxy Command
- ▶ Command Documentation
- ▶ Future CL Enhancements



Pointer variables

– V5R4

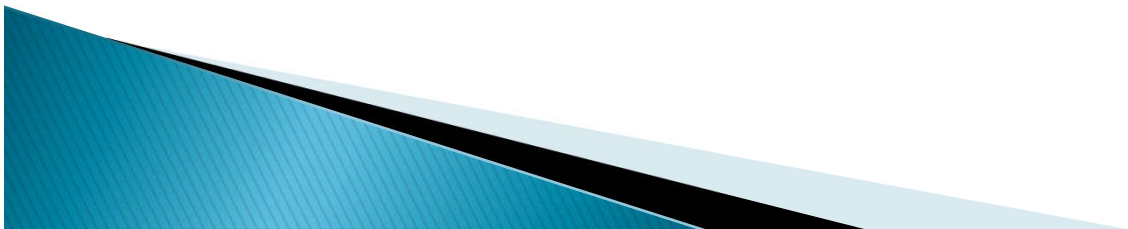
- ▶ Add TYPE(*PTR) on DCL statement
- ▶ New %ADDRESS built-in to set pointer
- ▶ New %OFFSET built-in to store pointer offset
- ▶ Add *BASED attribute on DCL statement
- ▶ Add *DEFINED attribute on DCL statement
- ▶ Allow pointer to be used with %SUBSTRING
- ▶ Makes many functions available to ILE CL
 - Full record-level file I/O
 - String functions



Pointer Variables

– V5R4

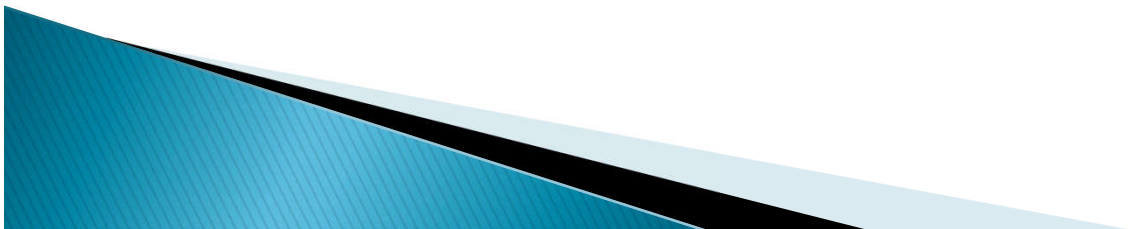
- ▶ New TYPE values on DCL statement
- ▶ Value
 - *PTR – Pointer
- ▶ DCL &SAMPLEPTR *PTR
 - Declares a pointer CL variable named &SAMPLEPTR which is a space pointer at the machine interface level
- ▶ DCL &CHARPTR *PTR ADDRESS(&CHAR)
 - Declares a pointer CL variable, &CHARPTR that is populated with the address of previously defined variable &CHAR
- ▶ Pointers are 16 bytes long
 - 128 Bit Address Space yields 16 Bytes.



*PTR Example with assignment

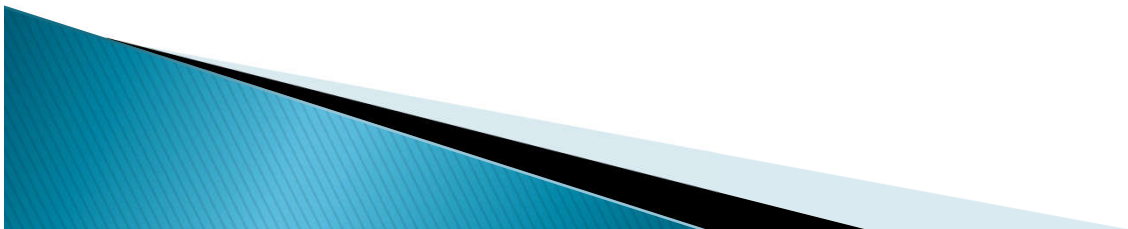
```
/* Character variable in Automatic Storage */  
DCL &CHAR *CHAR LEN(10)  
/* Pointer variable with address of &CHAR */  
DCL &PTR *PTR ADDRESS(&CHAR)
```

- ▶ The second DCL command declares a pointer variable which is initialized to point to the &CHAR variable in the program's automatic storage.



Based Variables – V5R4

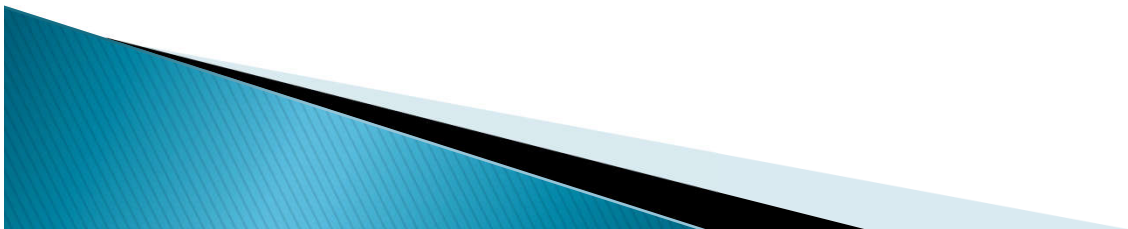
- ▶ Comprised of two new parms on DCL statement
- ▶ Parms:
 - STG(*BASED) (Storage is based on a pointer)
 - Default for this new parm is *AUTO for Automatic Storage
 - This is for compatibility with all previous OS versions
 - BASPTR(&PTR1) (Points to the storage for the variable.)
- ▶ DCL &CHAR1 *CHAR 10 STG(*BASED)
BASPTR(&PTR1)
 - Declares a 10-byte character CL variable named &CHAR1 that is based on the pointer CL variable &PTR1



*BASED Example

```
    / * A pointer variable */  
DCL  &PTR2  *PTR2  
    /* A variable based on the pointer variable above. */  
DCL  &CHAR2 *CHAR LEN(10) STG(*BASED)  
      BASPTR(&PTR2)
```

- ▶ The second DCL command declares a character variable which is found at the location addressed by the &PTR2 variable.
- ▶ Before &CHAR2 can be used, &PTR2 must be initialized to a valid address by using the %ADDRESS built-in function.



Defined Variables – V5R4

- ▶ Comprised of two new parms on DCL statement
- ▶ Parms:
 - STG(*DEFINED)
 - Storage is Defined within another var.
 - Requires the following:
 - DEFVAR(&CHAR3 3)
 - Part one defines the host variable this variable is defined inside of.
 - Part two designates the starting position within the host variable
- ▶ Effectively data structures and subfields for CL



Defined Variables – Example

```
/* Character variable in Automatic Storage */  
DCL &CHAR3 *CHAR LEN(100)
```

```
/* Defined variable hosted by above variable */  
DCL &DEC1 *DEC LEN(10 5) STG(*DEFINED)  
DEFVAR(&CHAR3 3)
```

- Declares a 10–digit (packed) decimal CL variable, &DEC1
- &DEC1 is hosted by &CHAR3 (which is in automatic storage)
- &DEC1 begins in position 3 of &CHAR3



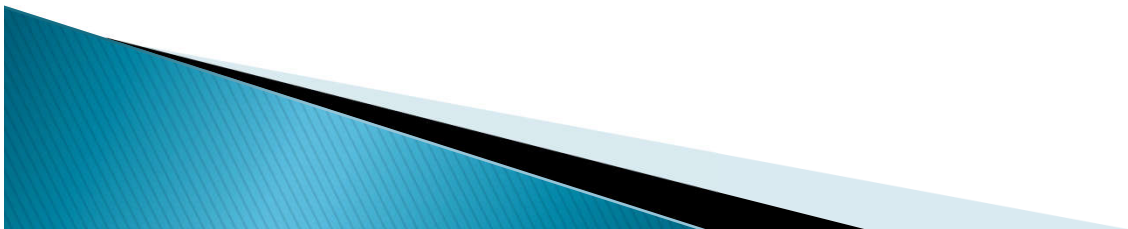
*DEFINED – USEFUL – Example

```
/* Fully Qualified Object Name (Also used as incoming PARM  
value) */  
DCL &QUALOBJ *CHAR LEN(20)
```

```
/* Object name only – Bytes 1–10 of fully qualified name */  
DCL &OBJ *CHAR LEN(10) STG(*DEFINED) DEFVAR(&QUALOBJ 1)
```

```
/* Library name only – Bytes 11–20 of fully qualified name */  
DCL &LIB *CHAR LEN(10) STG(*DEFINED) DEFVAR(&QUALOBJ 11)
```

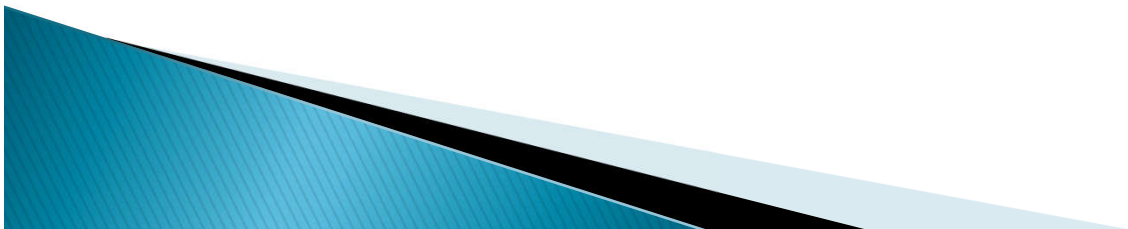
- ▶ The first DCL command declares a 20-character variable in the program's automatic storage.
- ▶ The second DCL command declares a variable named &OBJ which refers to the first 10 characters of the &QUALOBJ variable.
- ▶ The last DCL command declares a variable named &LIB which can be used to reference the last 10 characters of the &QUALOBJ variable.
- ▶ Very useful for situations where you are pulling apart a defined data structure!



*DEFINED *PTR Example

```
    / * Character variable */  
DCL  &CHAR4 *CHAR4 LEN(48)  
    /* Pointer variable defined in &CHAR4 */  
DCL  &PTR  *PTR  STG(*DEFINED)  
      DEFVAR(&CHAR4 17)
```

- ▶ The second DCL command declares a pointer variable in bytes 17 through 32 of the variable &CHAR4.
 - Pointers are 16 bytes long.
- ▶ Essentially this points out that it's not relevant which type of variable the hosted variable is.



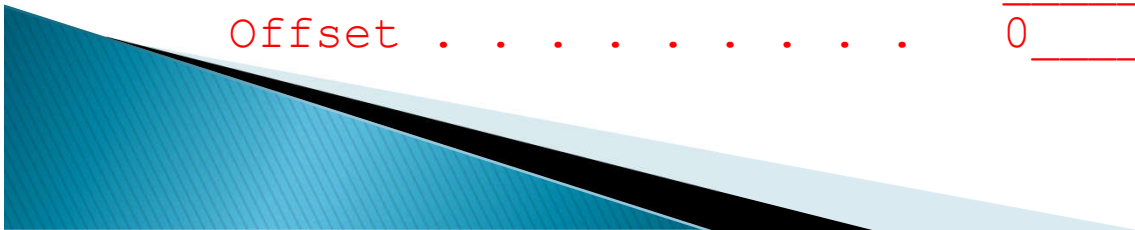
Declare CL Variable (DCL) (New)

Type choices, press Enter.

CL variable name	_____	Variable name
Type	_____	*DEC, *CHAR,
	*LGL, *INT, *UINT, *PTR	
Storage	*AUTO _____	*AUTO, *BASED,
		*DEFINED
Length of variable:		
Length	_____	Number
Decimal positions	_____	Number
Initial value	_____	

... .

Basing pointer variable	_____	Variable name
Defined on variable:		
CL variable name	_____	Variable name
Position	1 _____	1-32767
Address:		
CL variable name	_____	Variable name
Offset	0 _____	0-32766



%ADDRESS BIF Example

```
/* A pointer variable */  
DCL &PTR3 *PTR
```

```
/* A variable based on the pointer variable above. */  
DCL &CHAR5 *CHAR LEN(10) STG(*BASED) BASPTR(&PTR3)
```

```
/* A character variable in automatic storage */  
DCL &ACHAR *CHAR LEN(10)
```

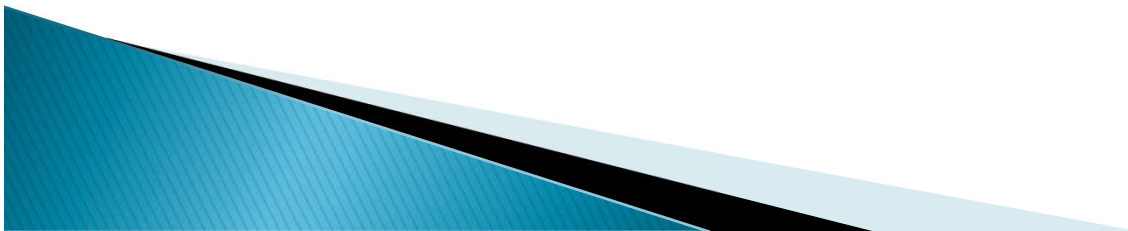
```
CHGVAR VAR(&PTR3) VALUE(%ADDRESS(&ACHAR))
```

- ▶ CHGVAR command places the address of &ACHAR into the pointer variable &PTR3
- ▶ References to variable &CHAR5 will reference the same storage as &ACHAR.



Support for Integer Variables – V5R3

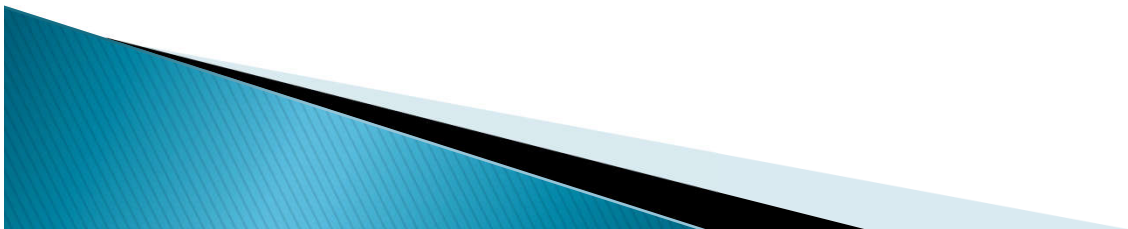
- ▶ Much "cleaner" than using %BIN
 - Use the value natively
- ▶ Useful for
 - passing parameters to IBM i APIs
 - passing parameters to other HLL programs
- ▶ Command PARM statement will allow RTNVAL(*YES) for integer parameters



Integer Variables

– V5R3

- ▶ New TYPE values on DCL statement
- ▶ Values
 - *INT – Integer
 - *UINT Unsigned Integer
 - chosen for consistency with PARM TYPE values
- ▶ LEN(2) and LEN(4) supported
- ▶ OPM does not fully support 8-byte integers internally so they cannot be supported in the language.



Integer Variables

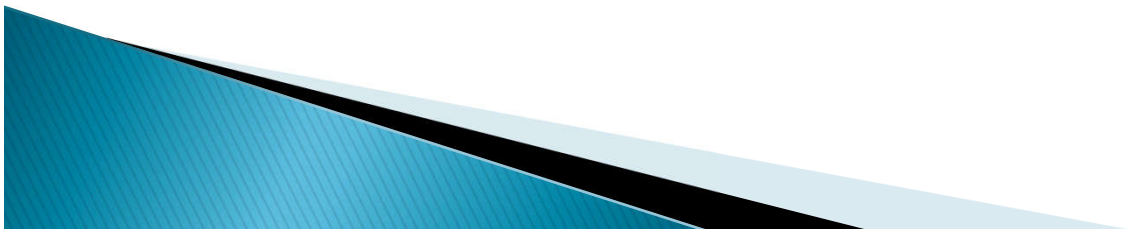
– IBM i 7.1

- ▶ New LEN(8) supported in CLLE
- ▶ Support for both types
 - *INT – Integer
 - *UINT Unsigned Integer
- ▶ As stated previously, OPM does not fully support 8-byte integers internally so no LEN(8) support coming there.
- ▶ This is important support for API calls as more and more are utilizing 8 byte support.
- ▶ For a very long time the help text for DCL claimed a max length of 4.
 - If your systems says this you need to get PTFs. 😊



Putting it together

- ▶ On the following slide we'll examine a sample program that puts together Pointers, Offsets, Based and Defined variables.
- ▶ Variable &VAR is a text variable of 500 characters.
- ▶ Variable &ARY is a text variable that is based on pointer &PTR and is 50 bytes long.
- ▶ Variables &BYT0110, &BYT1120 etc are defined as overlaying variable &ARY.
- ▶ Pointer &PTR is initialized to the first position of &VAR thus overlaying &ARY and the &BYTnnnn variables.
- ▶ In the loop the offset is incremented by 50 bytes thus giving us a view of each 50 bytes in the array.
- ▶ This technique is well used in parsing the data coming back from API calls in User Spaces.

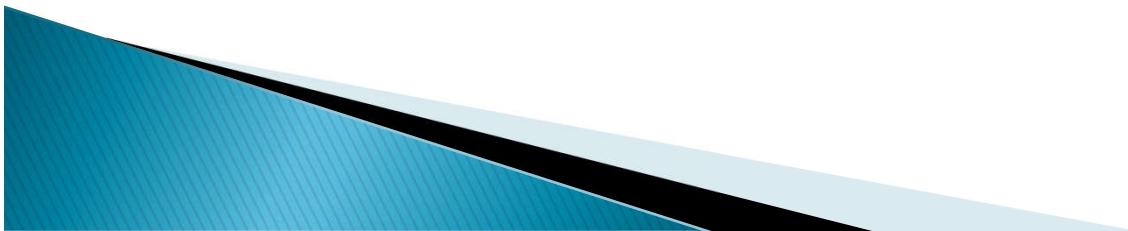


PGM

```
DCL          VAR(&VAR) TYPE(*CHAR) LEN(500) +
VALUE( a111456789j09a222456789j09a333456789j09a444456789j09a555456789j
...      j111456789j222456789j333456789j444456789j555456789`)
DCL &PTR          TYPE(*PTR)
DCL &ARY          TYPE(*CHAR) STG(*BASED)      LEN(50)  BASPTR(&PTR)
DCL &BYT0110     TYPE(*CHAR) STG(*DEFINED)    LEN(10)  DEFVAR(&ARY 01)
DCL &BYT4150     TYPE(*CHAR) STG(*DEFINED)    LEN(10)  DEFVAR(&ARY 41)
DCL &OFS         TYPE(*INT )                  LEN(4)   VALUE(1)
CHGVAR &PTR %ADDRESS(&VAR) /* Pointer points at var &VAR */
/* As a result &ARY now overlays first 50 bytes of &VAR */
CHGVAR &OFS %OFFSET(&PTR) /* Offset initialized to first byte */
DOFOR          VAR(&INT) FROM(1) TO(10) /* Actual string parse code
*/
CHGVAR &TEXT (&BYT0110 || '=' || &BYT1120 || '=' ||
&BYT2130 || '=' || &BYT3140 || '=' || &BYT4150)
SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG) MSGDTA(&TEXT) +
TOPGMQ(*EXT) MSGTYPE(*STATUS)
DLYJOB        DLY(2)
CHGVAR        &OFS (&OFS + 50)
CHGVAR        %OFFSET(&PTR) &OFS
ENDDO
ENDPGM
```

New special value – IBM i 6.1

- ▶ New special value *NULL
- ▶ Used for setting or testing pointer variables.
 - Example DCL &PTR *PTR ADDRESS(*NULL)
- ▶ IF (&PTR *EQ *NULL)
 - Test easily for a null pointer value preventing execution errors.



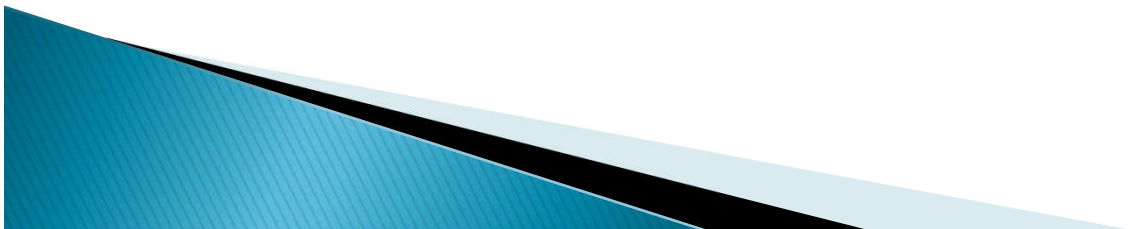
Increased size for *CHAR variables – V5R3

- ▶ Previous limit was 9999 bytes for CL variables declared as TYPE(*CHAR)
- ▶ New limit is 32767 bytes for TYPE(*CHAR)
- ▶ DCLF will not generate CL variables for character fields longer than 9999 bytes in a record format; same compile-time error
 - V5R4 Removed this limitation.
- ▶ Limit for TYPE(*CHAR) and TYPE(*PNAME) on PARM, ELEM, and QUAL command definition statements stays at 5000 bytes



Agenda:

- ▶ Variable Types
- ▶ Parameter enhancements
- ▶ Multiple File Support
- ▶ Declare Processing Options
- ▶ Source member Include
- ▶ Control Flow Enhancements
- ▶ Subroutines
- ▶ Command Enhancements
- ▶ New API QCAVFYNM
- ▶ Proxy Command
- ▶ Command Documentation
- ▶ Future CL Enhancements



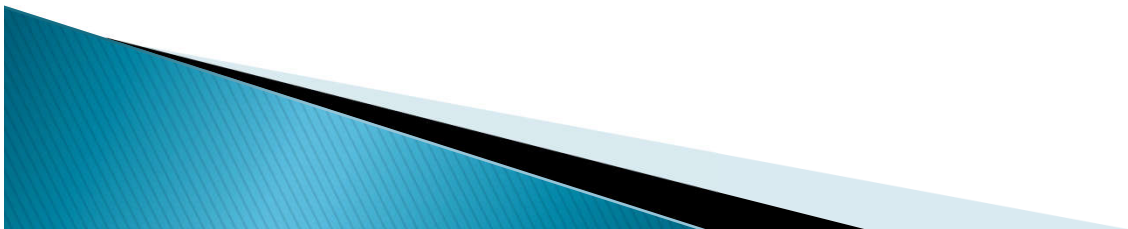
Parameter passing "by value" –V5R3

- ▶ CALLPRC (Call Procedure) command supports calls from ILE CL procedures to other ILE procedures
- ▶ In prior releases, CALLPRC only supported passing parameters "by reference"
- ▶ Can specify *BYREF or *BYVAL special value for each parameter being passed
 - CALLPRC PARM((&PARAM1 [*BYREF/*BYVAL]))
- ▶ Enables ILE CL to call many MI and C functions and other OS/400 procedure APIs
- ▶ Maximum numbers of parameters still 300



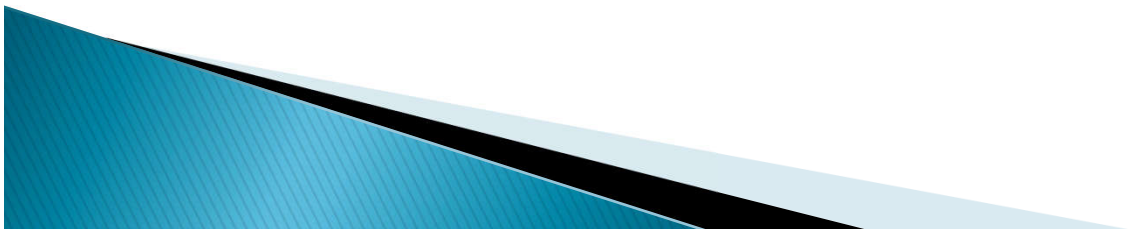
Increase max number of parms – V5R3

- ▶ Previous limit was 40 for PGM and TFRCTL, and 99 for CALL command
- ▶ New limit is 255 parameters for PGM, CALL, and TFRCTL
- ▶ Limit for CALLPRC (only allowed in ILE CL procedures) will stay at 300
- ▶ Number of PARM statements in a CL command will stay at 99



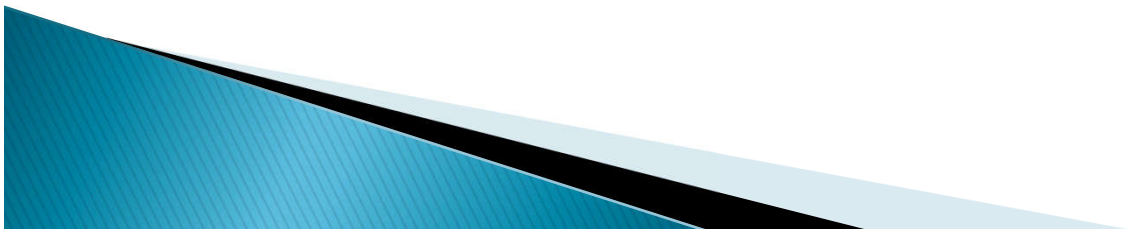
Agenda:

- ▶ Variable Types
- ▶ Parameter enhancements
- ▶ Multiple File Support
- ▶ Declare Processing Options
- ▶ Source member Include
- ▶ Control Flow Enhancements
- ▶ Subroutines
- ▶ Command Enhancements
- ▶ New API QCAVFYNM
- ▶ Proxy Command
- ▶ Command Documentation
- ▶ Future CL Enhancements



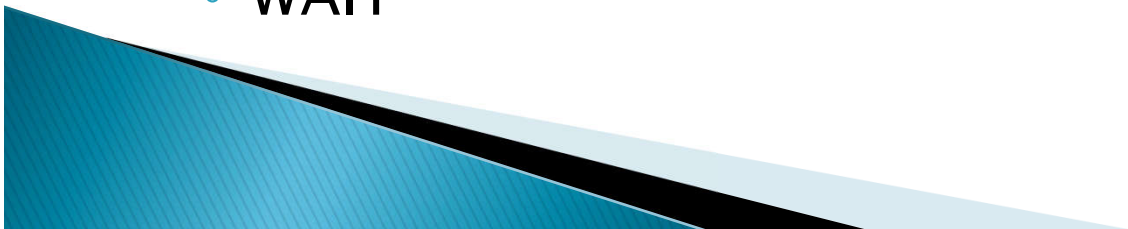
Multiple File Support – V5R3

- ▶ Supports up to 5 file "instances"
- ▶ Instances can be for the same file or different files
- ▶ New OPNID (Open identifier) parameter added to DCLF statement
- ▶ Default for OPNID is *NONE
 - Only one DCLF allowed with OPNID(*NONE)
- ▶ OPNID accepts 10-character name (*SNAME)
 - DCLF FILE(LIBA/FILE1) **OPNID(OPENIDENT5)**



Multiple File Support (continued)

- ▶ If OPNID name specified, declared CL variables are prefixed by this name and an underscore
- ▶ So FLDA is defined as **&OPENIDENT5_FLDA**
- ▶ OPNID also added to existing file input/output CL statements
 - RCVF
 - ENDRCV
 - SNDF
 - SNDRCVF
 - WAIT



CL6: PGM

DCLF FILE(OBJLST) OPNID(P1) /* NEW OPNID */

OPNID(P2)

OPNID(P1)

MONMSG MSGID(CPF0864) EXEC(GOTO CMDLBL(LOOP1B))
CHGVAR VAR(&COUNT) VALUE(&COUNT + 1)

P1_

LOOP1B: CHGVAR VAR(&TTOTSIZE) VALUE(&TTOTSIZE)

OPNID(P2)

MONMSG MSGID(CPF0864) EXEC(GOTO CMDLBL(LOOP2B))
CHGVAR VAR(&COUNT) VALUE(&COUNT + 1)
CHGVAR VAR(&TOBJSIZE) VALUE(&P2_ODOBSZ)
SNDPGMMSG MSGID(CPF9898) MSGF(QCPFMSG) ...
GOTO CMDLBL(LOOP2)

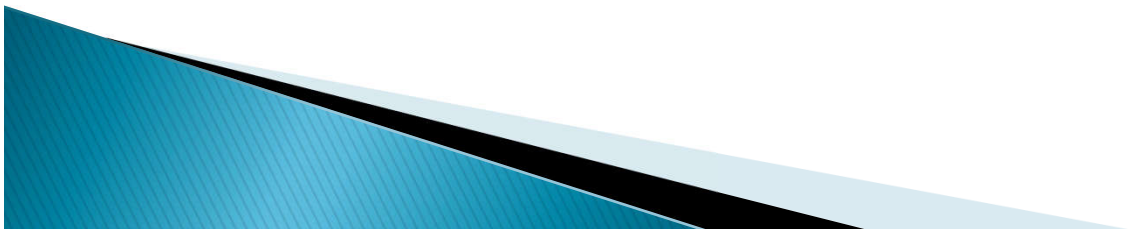
LOOP2B: ENDPGM

Multiple File Support Enhanced – IBM i 6.1

Syntax:

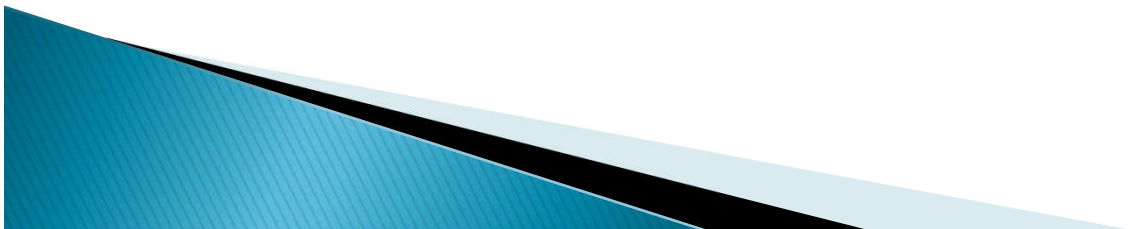
CLOSE OPNID(P1)

- ▶ New command CLOSE supports closing DB Files.
 - Single OPNID (Open identifier) parameter
 - Default for OPNID is *NONE (Consistency!)
- ▶ OPNID accepts 10-character name (*SNAME type)
- ▶ The next use of RCVF will implicitly reopen the file.
 - The record pointer will be reset to the same record it was the first time.
 - This USUALLY means the beginning of the file but if previously deleted records before that record are now occupied, they may not be read.



Agenda:

- ▶ Variable Types
- ▶ Parameter enhancements
- ▶ Multiple File Support
- ▶ Declare Processing Options
- ▶ Source member Include
- ▶ Control Flow Enhancements
- ▶ Subroutines
- ▶ Command Enhancements
- ▶ New API QCAVFYNM
- ▶ Proxy Command
- ▶ Command Documentation
- ▶ Future CL Enhancements



Declare Processing Options – V5R4

Syntax:

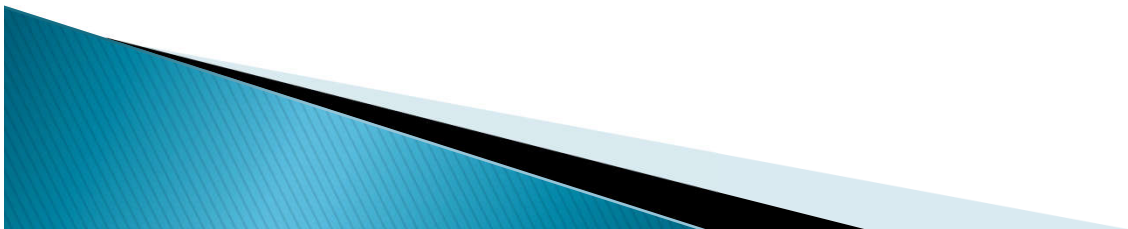
DCLPRCOPT SUBRSTACK(99)

- ▶ Indicates the maximum number of subroutine levels allowed at run time.
 - Min value is 20
 - Default is 99
 - Maximum is 9999
- ▶ Must be placed in the ‘DCL Section’ of the program (Before executables.)
- ▶ Only one per program.



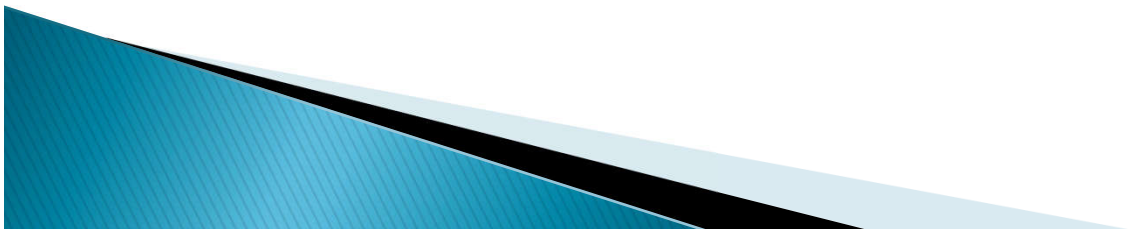
Declare Processing Options – Enhanced– IBM i 6.1

- ▶ Each of the additional parms override the corresponding parm of the CRTxxx CMD
- ▶ These parms have no defaults.
 - Allows the CRTxxx Defaults to work.
- ▶ Overrides shown on the compile printout.
- ▶ Not all parms apply to all CRTxxx CMDs
 - i.e. some for OPM only, some for ILE only.
- ▶ While we're talking about compiling, ILE programs can be compiled from IFS Source (i 7.3)
 - CRTCLMOD and CRTBNDCL



Declare Processing Options – Enhanced-IBM i 6.1

- ▶ LOG(*JOB *YES *NO)
 - LOG CL Commands.
- ▶ RTVCLSRC(*YES *NO) [OPM Only]
 - Allow retrieval of CL Source from compiled object.
- ▶ TEXT('description goes here' *SRCMBRTXT *BLANK)
 - Place this text on the compiled object.
- ▶ USRPRF(*USER *OWNER)
 - Specifies which profile to use during run-time for authority checking.
 - Ignored for REPLACE(*YES) on existing PGM
- ▶ AND MORE, Prompt DCLPRCOPT to see them all

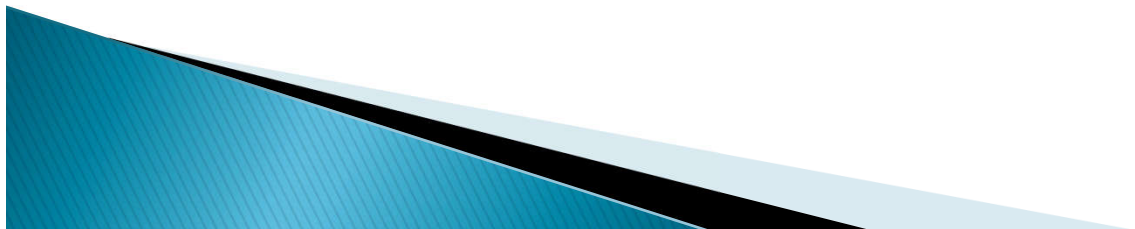


While we're talking about compiling..

.. A very brief interlude...

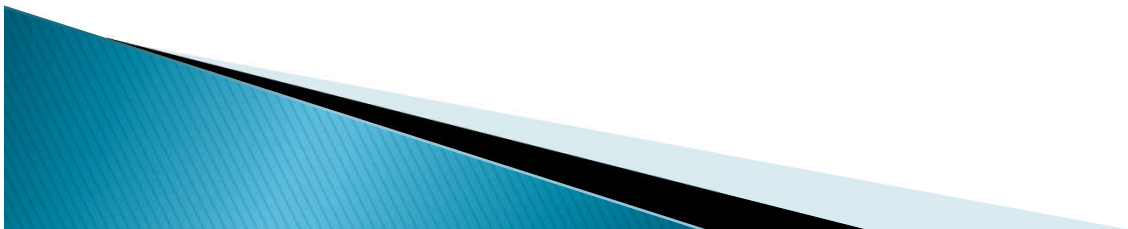
- ▶ ILE programs can be compiled from IFS Source
- ▶ Supports use of GIT et al for source storage
- ▶ Can be full path to source, or a file name only in which case the job's current working directory is appended.
- ▶ Mutually exclusive to SRCMBR and SRCFILE Params
- ▶ Support arrived in i 7.3
- ▶ Supported by CRTCLMOD and CRTBNDCL
- ▶ NOT supported by old form CL

Program	_____	Name
Library	*CURLIB	Name, *CURLIB
Source file	QCLSRC	Name
Library	*LIBL	Name, *LIBL, *CURLIB
Source member	*PGM	Name, *PGM
Source stream file	_____	




Declare Processing Options – Enhanced-IBM i 6.1

- ▶ **AUT(*LIBCRTAUT *CHANGE *ALL *USE *EXCLUDE autl)**
 - Specifies the authority to users who do not have any explicit authority to the object.
 - Ignored for REPLACE(*YES) on existing PGM
- ▶ **SRTSEQ(*HEX *JOB *JOB RUN...) or (lib/obj)**
 - Specifies the sort sequence to use for the job.
 - Details on the command ☺
- ▶ **LANGID(*JOB RUN *JOB language-ID)**
 - Language ID to use for the job.
- ▶ **STGMDL(*SNGLVL *TERASPACE) [CRTBNDCL only]**
 - *SNGLVL runs only in a single-level storage activation group
 - *TERASPACE runs only in a teraspace activation group.
 - DFTACTGRP(*YES) NOT allowed with *TERASPACE

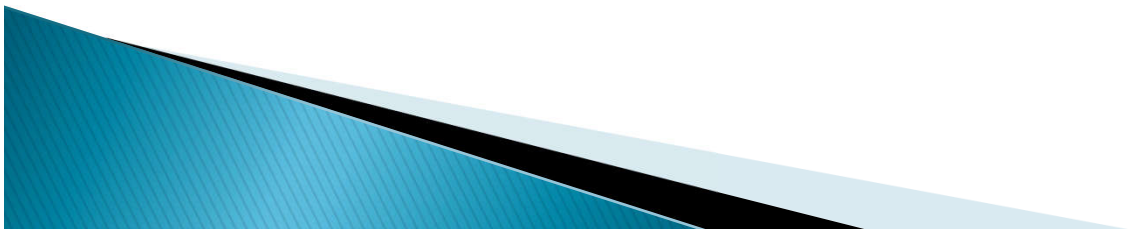


Declare Processing Options – Enhanced– IBM i 6.1

- ▶ **DFTACTGRP(*YES *NO)** [CRTBNDCL only]
 - Specifies if the program is associated with the default activation group.
 - ▶ **ACTGRP(*STGMDL *CALLER *NEW)** [ILE CL]
 - Specifies the activation group that the ILE CL program runs in.
 - ▶ **BNDSRVPGM(library/name Generic_name *ALL)**
 - Specifies the service program or programs to search for unresolved module requests at bind time.
 - ▶ **BNDDIR(*NONE) or (library/directory)**
[CRTBNDCL only]
 - Specifies the list of binding directories used in symbol resolution.
 - Used only if unresolved imports exist after modules and service programs are considered.
- 

Agenda:

- ▶ Variable Types
- ▶ Parameter enhancements
- ▶ Multiple File Support
- ▶ Declare Processing Options
- ▶ Source member Include
- ▶ Control Flow Enhancements
- ▶ Subroutines
- ▶ Command Enhancements
- ▶ New API QCAVFYNM
- ▶ Proxy Command
- ▶ Command Documentation
- ▶ Future CL Enhancements

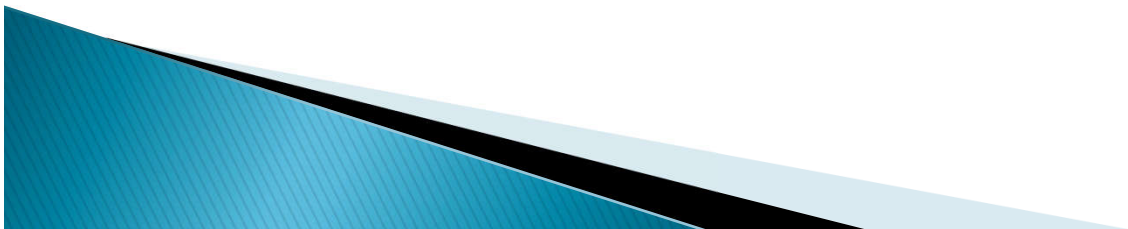


Source INCLUDE – IBM i 6.1

Syntax:

```
INCLUDE SRCMBR(ANINCLUDE)  
  SRCFILE(library/file)
```

- ▶ Defines a source member to include at compile time.
- ▶ SRCMBR Parm defines the source member to include (required)
- ▶ SRCFILE Defaults to *SRCFILE
 - *SRCFILE default is the file *this* CL program is in.
- ▶ Initially INCLUDE was not allowed within an Included source, that is, no nesting.



Source INCLUDE – IBM i 6.1

Syntax addition for compile commands:

INCFILE(library/file)

- ▶ INCFILE Default is *SRCFILE
 - Indicates the include members are found in the same source file as the CL source member being compiled.
- ▶ Specifying a file and optionally a library overrides the file for any INCLUDE specifying *SRCFILE
- ▶ CRTCLPGM, CRTCLMOD and CRTBNDCL all support this parm.



INCLUDE – Additional Details – IBM i 6.1

Retrieve CL Source enhanced to optionally retrieve the included source.

RTVINCSRC(*YES, *NO)

- ▶ Default is *NO
- ▶ Specifying *YES will generate source that has the included source embedded into it.
 - The INCLUDE line is NOT regenerated, rather the included source represents what was compiled.
- ▶ Specifying *NO will include the original INCLUDE command in the retrieved source



Nested INCLUDEs – IBM I 7.1

- ▶ INCLUDE will be supported within INCLUDE members.
- ▶ No limit to the number of includes (in the O/S anyway) YOU may go crazy if they go too deep!
- ▶ I created a trivial CLLE Program that included itself. (Note: This is bad practice! 😊)
 - The compile took about 2 minutes to fail with an MCH2804: “Tried to go larger than storage limit for object.”
 - Followed by CPF2524 RC 5: “the exception handler for an exception was an internal procedure that was already busy handling a previous exception. ”



While we're talking about includes.

.. Another interlude...

- ▶ As ILE programs can be now compiled from IFS Source, the INCLUDE statement can also reference IFS source.
- ▶ Also arrived in i 7.3
- ▶ Also supported by CRTCLMOD and CRTBNDCL only.
 - These commands gain new OPTIONAL parameter INCDIR.
 - Up to 32 directories may be specified .

```
INCLUDE      SRCSTMF('AnIncludeFileHere.cl')
```

- This full path form ignores INCDIR
- Default is the path the source member is found in.

```
INCLUDE      SRCSTMF('/home/ldb/anninclude.cl')
```

- ▶ NOT supported by old form CL
 - You can key it in but the compile will fail CPD0043



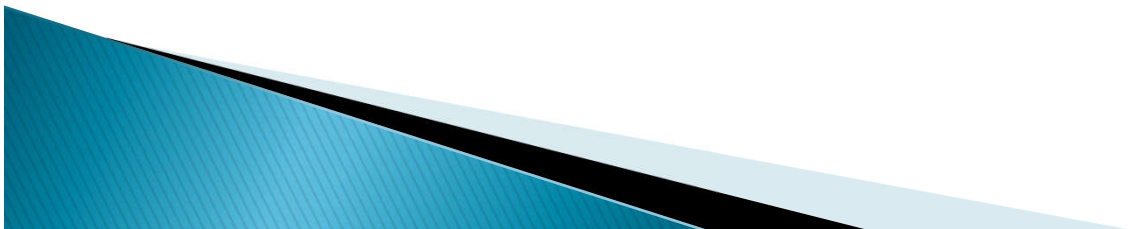
Retrieving CL Source – IBM I 7.1

- ▶ Speaking of retrieving source from CL Programs!
- ▶ Support includes
 - *MODULE
 - *PGM
 - *SRVPGM
- ▶ CRTCLMOD and CRTBNDCL commands get new parm.
 - ALWRTVCLSRC
 - Default *YES as it is for CRTCLPGM



Agenda:

- ▶ Variable Types
- ▶ Parameter enhancements
- ▶ Multiple File Support
- ▶ Declare Processing Options
- ▶ Source member Include
- ▶ Control Flow Enhancements
- ▶ Subroutines
- ▶ Command Enhancements
- ▶ New API QCAVFYNM
- ▶ Proxy Command
- ▶ Command Documentation
- ▶ Future CL Enhancements



Control Flow Enhancements – V5R3

Additional ‘standard’ control flow commands:

- ▶ DOWHILE, DOUNTIL, DOFOR

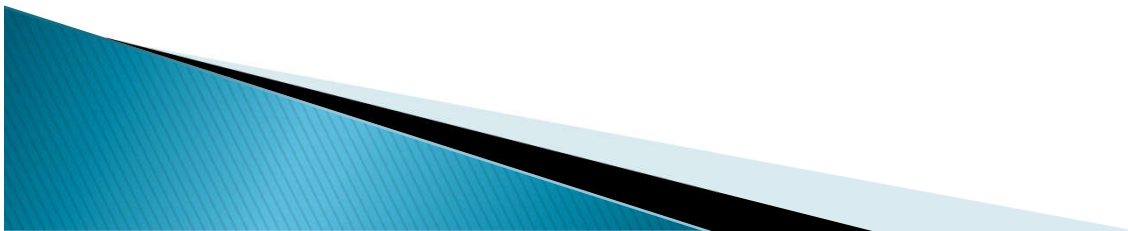
Each support

- LEAVE
- ITERATE

- ▶ CASE

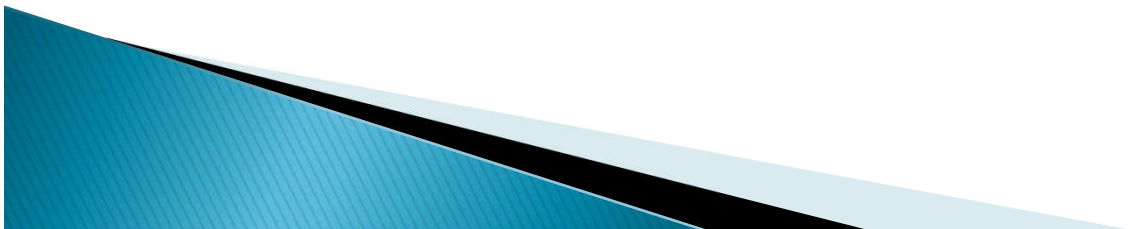
SELECT, WHEN, OTHERWISE, ENDSELECT

25 level nesting



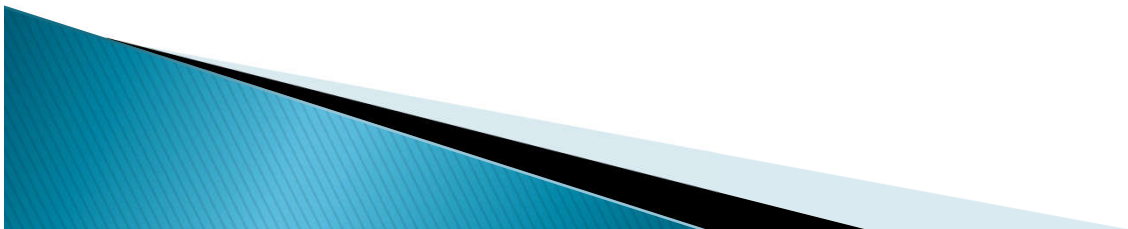
Common DOxxx Support Elements – V5R3

- ▶ Loop starts with the DOxxx statement
 - The DOxxx statement supports a label (note this)
- ▶ ENDDO marks end of loop
 - All types of DO loop use ENDDO
- ▶ ITERATE – Discontinue processing remainder of code before ENDDO and transfer to label on DOxxx
 - Can be the label on the current DOxxx or loops external to this loop
 - If no label given the current DOxxx loop is assumed



Common DOxxx Support

- ▶ LEAVE – Discontinue processing remainder of loop and jump to statement following the matching ENDDO
 - Can be the label on the DOxxx or the DOxxx loops external to this loop
 - If no label given the current DOxxx loop is assumed
- ▶ Can be nested (up to 25 levels)
 - i.e. you could have a DOWHILE loop within a DOFOR loop
 - or a DOWHILE inside a DOWHILE etc.



DOWHILE Loop – V5R3

- ▶ Same COND support as IF statement in CL
- ▶ Evaluates COND at "top" of loop
- ▶ A simple example:

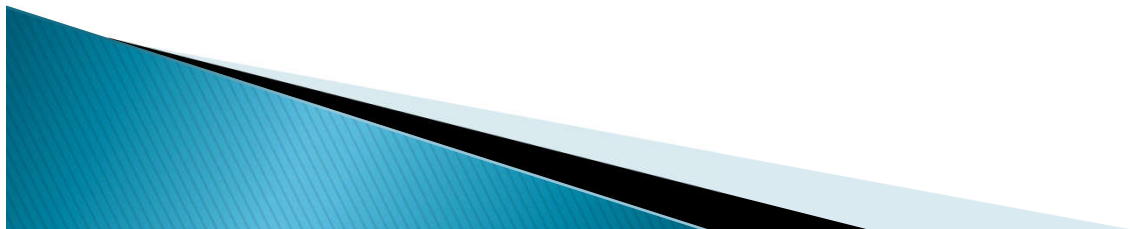
```
DCL VAR(&LGL) TYPE(*LGL) VALUE('1')
```

```
:
```

```
DOWHILE COND(&LGL)
```

```
: (group of CL commands)
```

```
ENDDO
```



DOUNTIL Loop – V5R3

- ▶ Same COND support as IF statement in CL
- ▶ Evaluates COND at "bottom" of loop
- ▶ A simple example:

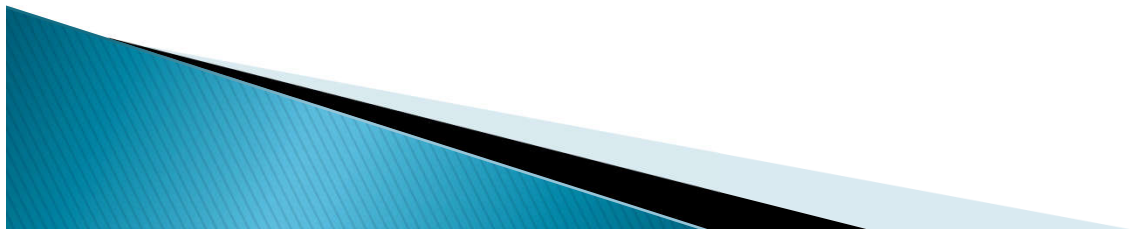
```
DCL VAR(&LGL) TYPE(*LGL) VALUE('0')
```

```
:
```

```
DOUNTIL COND(&LGL)
```

```
: (group of CL commands)
```

```
ENDDO
```

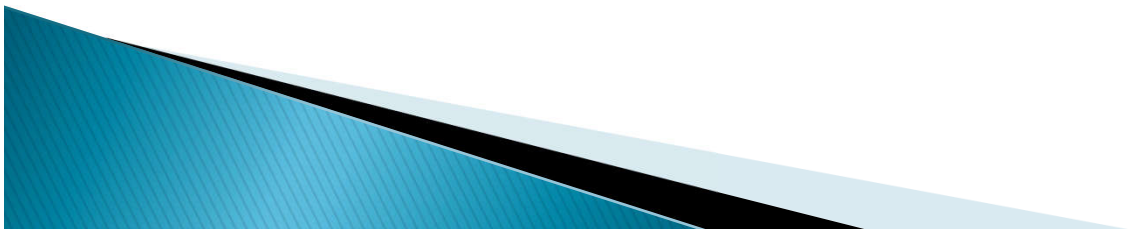


DOFOR Loop – V5R3

Syntax:

DOFOR VAR() FROM() TO() BY()

- ▶ BY defaults to '1', other parameters are required
- ▶ VAR must be *INT or *UINT variable
- ▶ FROM and TO can be integer constants, expressions, or variables
- ▶ BY must be an integer constant (can be negative)
- ▶ FROM/TO expressions are evaluated at loop initiation; TO evaluated after increment
- ▶ Checks for loop exit at "top" of loop



LEAVE and ITERATE – V5R3

- ▶ Allowed only within a DOWHILE, DOUNTIL or DOFOR group
- ▶ Both support LABEL to allow jump out of multiple (nested) loops
- ▶ Both default to *CURRENT loop
- ▶ LEAVE passes control to next CL statement following loop ENDDO
- ▶ ITERATE passes control to end of loop and tests loop exit condition

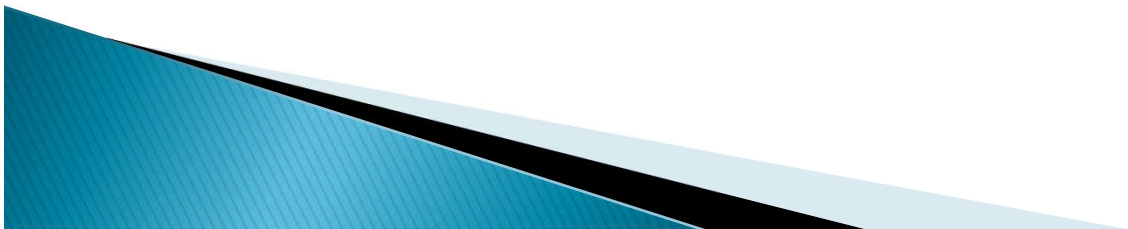
TAG: DOXXX

ITERATE TAG

LEAVE TAG

ENDDO /* Iterate transfer here */

/* Leave would transfer here */



LEAVE and ITERATE – Nested Example

```
LP1: DOUNTIL  &FLAG1=0  
LP2:          DOWHILE  &FLAG2=1  
LP3:          DOFOR  &COUNT FROM(1) TO(10)  
          BY(2)
```

```
LEAVE /* Jumps to (a) */
```

```
LEAVE LP1 /* Jumps to (c) */
```

```
ITERATE LP2 /* Jumps to (b) */
```

```
ENDDO /* End of DOFOR */
```

```
(a) (b) ENDDO /* End of DOWHILE */
```

```
ENDDO /* End of DOUNTIL */
```

```
(c) /* Statement after ENDDO */
```



SELECT Group – V5R3

- ▶ SELECT starts a group; this command has no parameters
- ▶ There must be at least one WHEN clause
 - Has COND and THEN support (like IF)
 - To execute multiple statements must use DO/ENDDO
 - Unlimited number of WHEN clauses may exist
- ▶ There may optionally be one OTHERWISE
 - Run if no WHEN statement COND = True
 - Single parm of CMD (like ELSE)
 - Again needs DO/ENDDO for multiple statements
- ▶ ENDSELECT ends group; this command has no parameters



SELECT Example

```
SELECT /* Begin of select group */
```

```
  WHEN COND((&COUNT *EQ 4) *AND (&COUNT2 *EQ 2)) THEN(DO)  
    ..some important stuff...  
  ENDDO
```

```
  WHEN COND(&COUNT *EQ 6) THEN(DO)  
    ..different important stuff..  
  ENDDO
```

```
  WHEN COND(&COUNT *EQ 3.141592654) THEN(CALLSUBR DESERT)
```

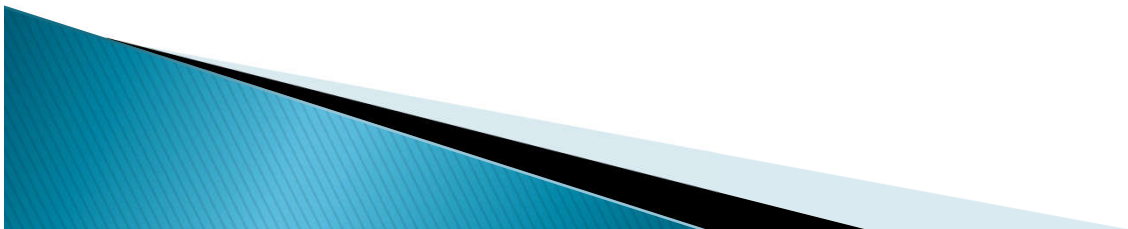
```
  OTHERWISE CMD(DO) /* OTHERWISE is optional */  
    ..default stuff..  
  ENDDO
```

```
ENDSELECT /* End of select group */
```



Control Flow – IBM i 7.1

- ▶ Select group indent on compile printouts.
- ▶ New value *DOSLTLVL for the OPTION() parameter on:
 - CRTCLPGM
 - CRTCLMOD
 - CRTBNDCL
 - This new parm tells the compiler to add a new column on the left with the nesting level.
- ▶ Default is *NODOSLTLVL which is same as today.
- ▶ Supports DO, DOFOR, DOUNTIL, DOWHILE and SELECT



Agenda:

- ▶ Variable Types
- ▶ Parameter enhancements
- ▶ Multiple File Support
- ▶ Declare Processing Options
- ▶ Source member Include
- ▶ Control Flow Enhancements
- ▶ Subroutines
- ▶ New BIFs
- ▶ Command Enhancements
- ▶ New API QCAVFYNM
- ▶ Proxy Command
- ▶ Command Documentation
- ▶ Future CL Enhancements



Subroutines! – V5R4

- ▶ All variables are global.
 - DCL* not allowed within a SUBR/ENDSUBR pair
- ▶ Recursion Allowed? YES!
 - Tried that. It works!
- ▶ Four Components
 - SUBR
 - Begin of Subroutine Definition
 - ENDSUBR
 - End of Subroutine Definition
 - CALLSUBR
 - Call a Subroutine
 - RTNSUBR
 - Return from a Subroutine

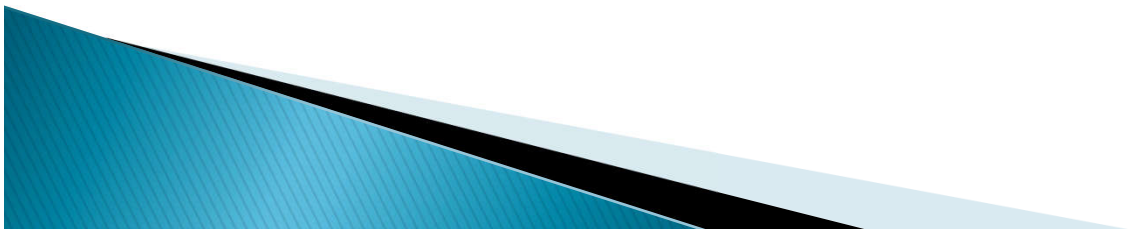


Subroutines – V5R4

Defines the beginning of the subroutine

SUBR SUBR(subroutine_name)

- ▶ A tag is optional.
 - May not be used to get *into* the subroutine
 - Used only to return to it's beginning from within it.
 - (you know with um, er, GOTO)
- ▶ SUBR cannot be between another SUBR/ENDSUBR pair (no nesting of definitions)

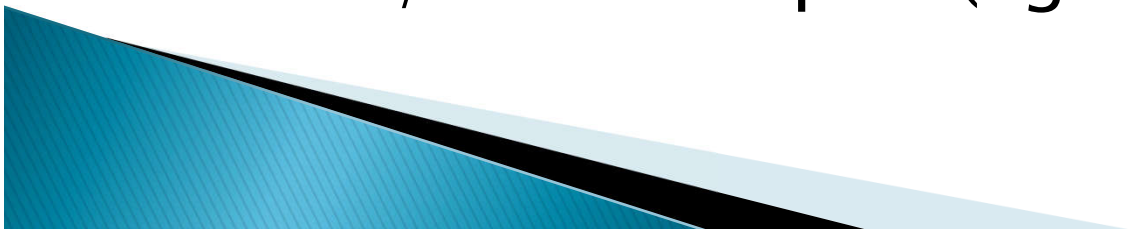


End of Subroutine – V5R4

Defines the end of the subroutine

ENDSUBR RTNVAL(return_var)

- Optional variable must be *INT of LEN(4)
- Can also return a constant
- Value is returned to caller such as error code.
- ▶ When execution reaches ENDSUBR execution passes to the statement following the CALLSUBR that invoked this subroutine
- ▶ ENDSUBR cannot be between another SUBR/ENDSUBR pair (again no nesting)

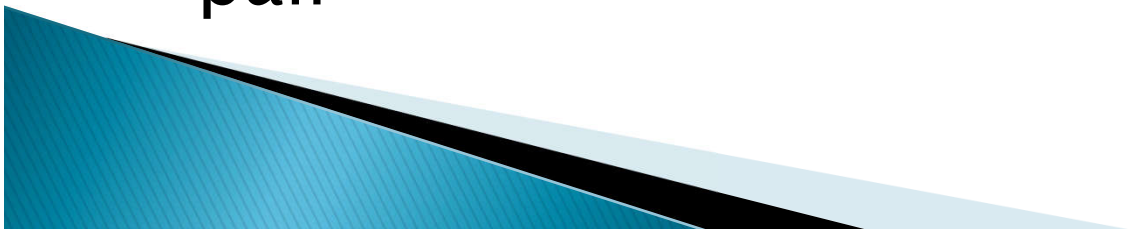


Return from Subroutine – V5R4

Defines another return from subroutine point

RTNSUBR RTNVAL(return_var)

- Optional variable must be *INT of LEN(4)
- Can also return a constant
- Value is returned to caller such as error code.
- ▶ Upon execution of RTNSUBR execution passes to the statement following the CALLSUBR that invoked this subroutine
- ▶ RTNSUBR Must be between SUBR/ENDSUBR pair



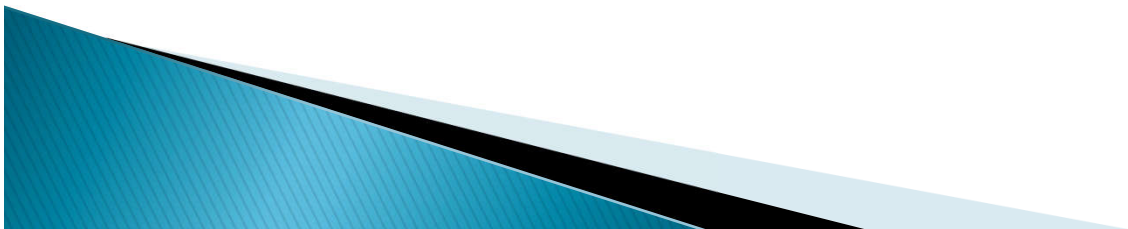
Call Subroutine – V5R4

Call a subroutine

`CALLSUBR SUBR(subroutine_name)`

`RTNVAL(return_var)`

- Optional RTNVAL variable must be *INT of LEN(4)
- Value is return only NOT passed into subroutine.
- ▶ May be between SUBR/ENDSUBR pair



```
SUBR:   PGM
      DCL  &SIGNINT  *INT      /* Regular Signed Integer */
```

```
      DOWHILE      COND (&SIGNINT < 100)
```

```
      DLYJOB 2
      ENDDO
```

```
      RTNSUBR      RTNVAL (&SIGNINT) /* Return from here */
      ENDDO
      CHGVAR &SIGNINT (&SIGNINT + 10)
      CALLSUBR      SUBR (SUBR1) RTNVAR (&SIGNINT)
      ENDSUBR      RTNVAL (&SIGNINT) /* End of the subroutine */
      DAEND: ENDPGM
```

Agenda:

- ▶ Variable Types
- ▶ Parameter enhancements
- ▶ Multiple File Support
- ▶ Declare Processing Options
- ▶ Source member Include
- ▶ Control Flow Enhancements
- ▶ Subroutines
- ▶ New BIFs
- ▶ Command Enhancements
- ▶ New API QCAVFYNM
- ▶ Proxy Command
- ▶ Command Documentation
- ▶ Future CL Enhancements



%TRIM – IBM i 7.1 SI49061

- ▶ Six new BIFS are provided by this PTF, 3 are:
 - %TRIM – Trim from both ends
 - %TRIML – Trim from Left (leading) end.
 - %TRIMR – Trim from Right (trailing) end.

Each has Two parms.

- 1) Variable to Trim
- 2) Character(s) to Trim

%TRIM(&VAR) – Trim Spaces (default)
%TRIMR(&VAR ‘*.’) – Trim Splats and periods.
%TRIML(&VAR &CHARS) – Trim what's in &CHARS



Sample output.

```
DCL &VAR *CHAR 40 ' This is a Text Variable. '
```

```
SNDMSG MSG('' || %TRIM(&VAR) || ''')
```

- ▶ "This is a Text Variable."

```
SNDMSG MSG('' || %TRIML(&VAR) || ''')
```

- ▶ "This is a Text Variable. "

```
SNDMSG MSG('' || %TRIMR(&VAR '._') || ''') [<-.space]
```

- ▶ " This is a Text Variable " [spaces and . Trimmed]



Remaining new BIFs – IBM i 7.1 SI49061

- ▶ Remaining new BIFS are:
 - %CHECK – Check Characters
 - %CHECKR – Check Characters from right (trailing) end
 - %SCAN – Scan for Characters

Each has three parameters.

- 1) Character(s) to look for (Comparator)
- 2) Character(s) to look at (Base-String)
- 3) Starting Position (Optional)

Each returns a numeric value.

A non-zero indicates position.

A zero indicates none found.



%CHECK – Check from left

Returns first position of base string that contains a character that does NOT appear in comparator string.

```
DCL &Str *CHAR 27 'ABCDEFGHIJKLMNO_PQRSTUVWXYZ'
```

```
DCL &Srch3 *CHAR 3 'MNO'
```

```
DCL &Srch5 *CHAR 5 'MNO '
```

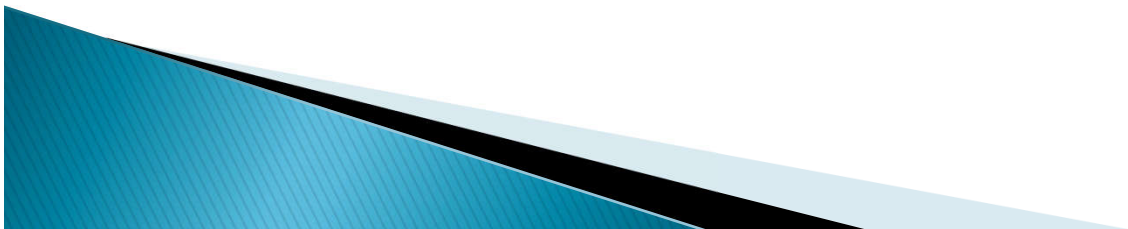
```
DCL &StPos *UINT 2 13
```

```
%CHECK(&Srch3 &Str &StPos) = 16 (space)
```

```
%CHECK(&Srch5 &Str &13) = 17 (P) Spaces count!
```

```
%CHECK(&Srch3 &Str ) = 1 (A)
```

```
%CHECK('MNO' &Str 14 ) = 16
```



%CHECKR – Check from right

Returns last position of base string that contains a character that does NOT appear in comparator string.

```
DCL &Str *CHAR 27 'ABCDEFGHIJKLMNO_PQRSTUVWXYZ'
```

```
DCL &Srch3 *CHAR 3 'MNO'
```

```
DCL &Srch5 *CHAR 5 'MNO '
```

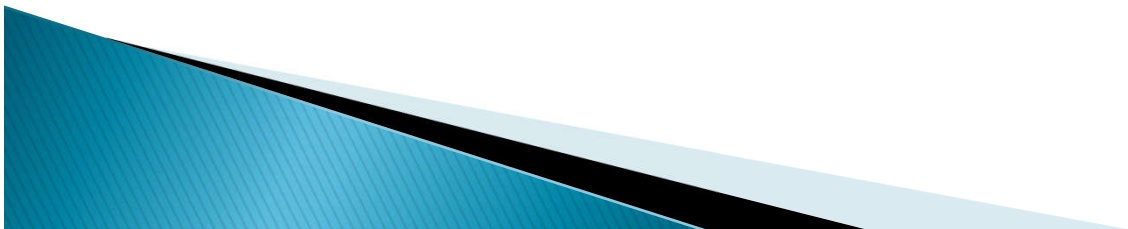
```
DCL &StPos *UINT 2 13
```

```
%CHECKR(&Srch3 &Str &StPos) = 12 (L)
```

```
%CHECKR(&Srch5 &Str ) = 27 (Z)
```

```
%CHECKR(&Srch3 &Str ) = 27 (Z)
```

```
%CHECKR('MNO' &Str 16 ) = 12 (L)
```



%SCAN – Scan for Characters

Returns position in base string that contains first character of comparator string.

```
DCL &Str *CHAR 27 'ABCDEFGHIJKLMNO_PQRSTUVWXYZ'
```

```
DCL &Srch3 *CHAR 3 'MNO'
```

```
DCL &Srch5 *CHAR 5 'MNO '
```

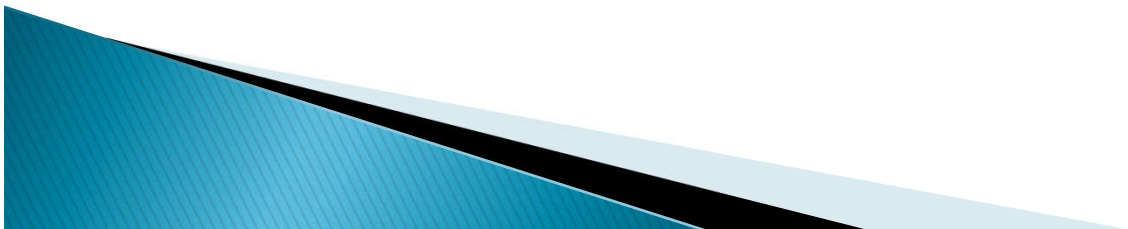
```
DCL &StPos *UINT 2 13
```

```
%SCAN(&Srch3 &Str &StPos) = 13 (MNO)
```

```
%SCAN(&Srch5 &Str ) = 0 (Not Found)
```

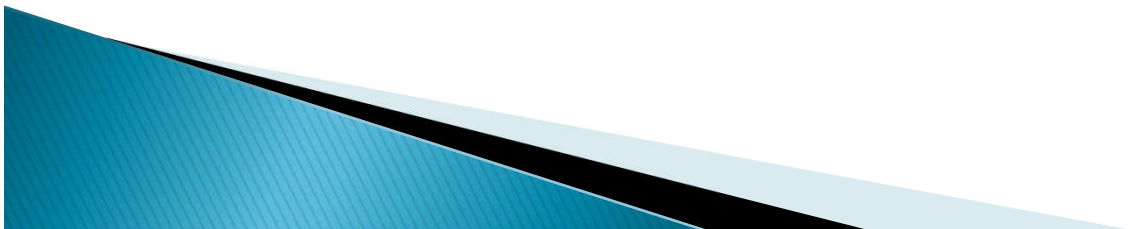
```
%SCAN(&Srch3 &Str ) = 13 (MNO)
```

```
%SCAN('MNO' &Str 14 ) = 0 (Not Found)
```



New BIFs, Rules

- ▶ **%TRIM, %TRIMR, %TRIML**
 - Valid anywhere a text variable is valid.
 - If any trim results in nothing, a full string of blanks is returned.
 - Second parm default is ' ' (spaces)
- ▶ **%CHECK, %CHECKR, %SCAN**
 - Valid anywhere a numeric variable is valid.
 - Starting Position is optional and defaults to 1.
 - %TRIM/R/L NOT Valid inside %CHECK/ %SCAN
 - %SCAN(%TRIM(&Srch) &Str &StPos) INVALID!
- ▶ **All work in CL, CLLE, and CL Modules**
- ▶ **CAN Compile back to (but not ON):**
 - IBM i 6.1
 - IBM i 5.4



New in 7.2 Conversion BIFs

- ▶ Convert to character format

`%CHAR(convert-argument)`

- The *convert-argument* must be a CL variable with TYPE of *LGL, *DEC, *INT or *UINT.
- For logical data, the result will be ether '0' or '1'.

- ▶ Convert to Decimal format

`%DEC(convert-argument [total-digits decimal-places])`

- The *convert-argument* must be a CL variable with TYPE of *CHAR, *LGL, *DEC, *INT or *UINT.

- ▶ Convert to Integer Format

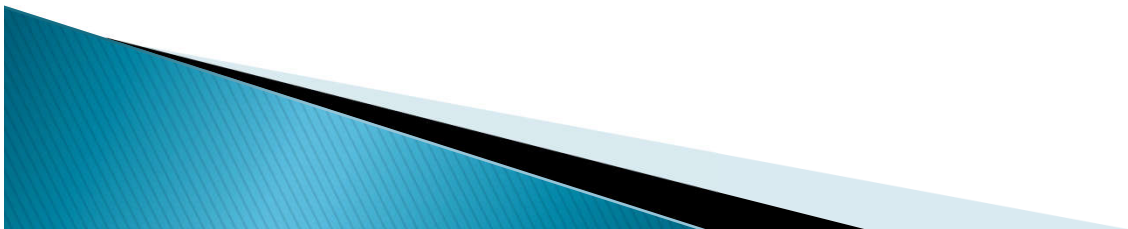
`%INT(convert-argument)`

- The *convert-argument* must be a CL variable with TYPE of *CHAR, *LGL, *DEC or *UINT.



New in 7.2 Conversion BIFs P2

- ▶ Convert to Unsigned Integer
%UINT(convert-argument)
%UNS(convert-argument)
 - The *convert-argument* must be a CL variable with TYPE of *CHAR, *LGL, *DEC or *INT.
- ▶ Convert string to lower case.
%LOWER(input-string [CCSID])
 - The *input-string* must be a CL variable with TYPE of *CHAR.
- ▶ Convert string to upper case.
%UPPER(input-string [CCSID])
 - The *input-string* must be a CL variable with TYPE of *CHAR.



i 7.2 BIFs for Size operations

- ▶ Return Length of a variable

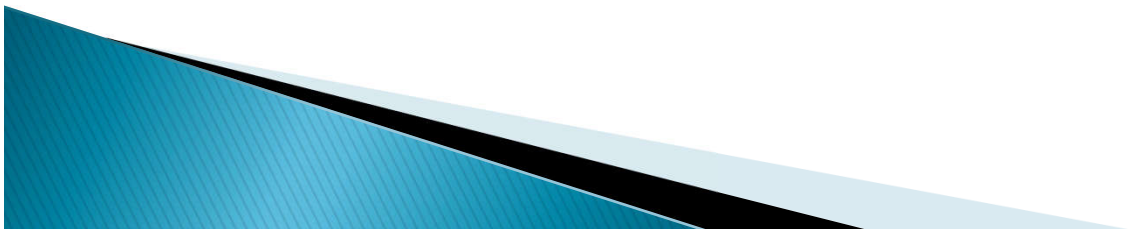
`%LEN(variable-argument)`

- The *variable-argument* must be a CL variable with TYPE of *CHAR, *DEC, *INT or *UINT.

- ▶ Return the number of bytes occupied by the CL Variable

`%SIZE(variable-argument)`

- The *variable-argument* must be a CL variable.



Agenda:

- ▶ Variable Types
- ▶ Parameter enhancements
- ▶ Multiple File Support
- ▶ Declare Processing Options
- ▶ Source member Include
- ▶ Control Flow Enhancements
- ▶ Subroutines
- ▶ New BIFs
- ▶ Command Enhancements
- ▶ New API QCAVFYNM
- ▶ Proxy Command
- ▶ Command Documentation
- ▶ Future CL Enhancements



Dynamic prompt messages – IBM i 6.1

- ▶ *CMD objects can now retrieve prompt text from message members
- ▶ CMD definition enhanced.
 - PROMPT parm can be text or MSGID
 - If MSGID new PMTFILE parm determines where to look for the message text.
 - Additional *STATIC or *DYNAMIC parm determines if prompt text lookup is done at compile time or run time.

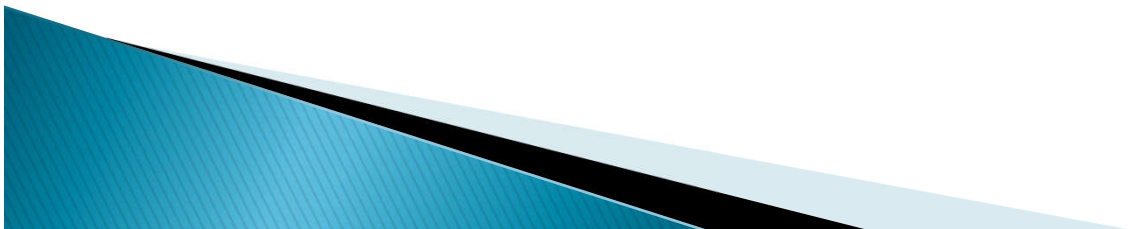
*NOTE

- ▶ Beginning with IBM i 6.1 this capability is used for all command objects
- ▶ The result of this is that from i 6.1 forward the QSYS29nn libraries containing only language specific commands were removed.
 - Security improvement!!



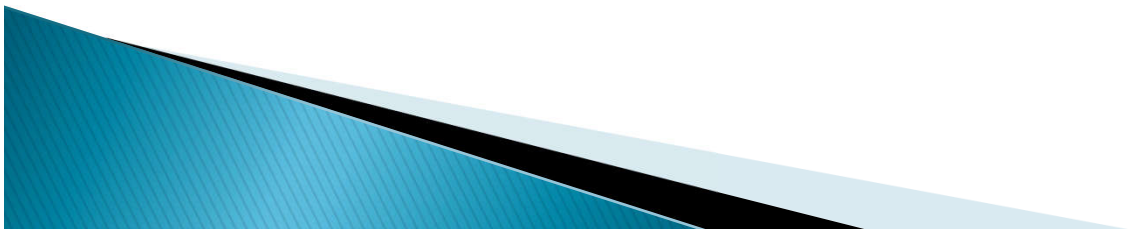
CMD definition enhancements – IBM i 6.1

- ▶ CMD definition to pull into the source many parms which currently must be specified on the CRTCMD
 - MAXPOS(0–99 *NOMAX)
 - Maximum Positional Parameters
 - ALLOW(*INTERACT *BATCH ...)
 - Where allowed to Run
 - MODE(*ALL *PROD ...)
 - Mode in which valid
- ▶ Pretty much all the parms from CRTCMD
- ▶ Too many to list here! (Press F4!)
 - But NOT the command processing program! ☹



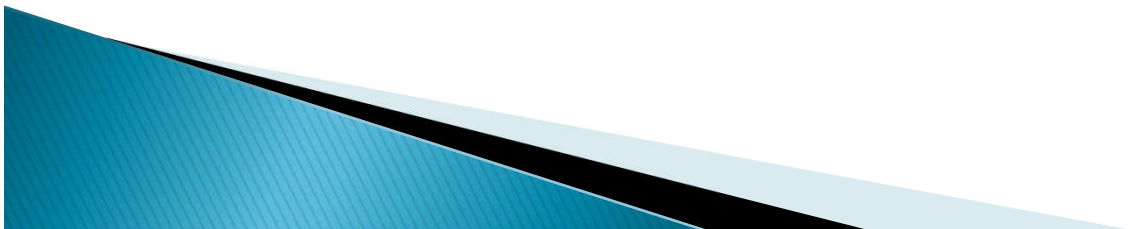
Agenda:

- ▶ Variable Types
- ▶ Parameter enhancements
- ▶ Multiple File Support
- ▶ Declare Processing Options
- ▶ Source member Include
- ▶ Control Flow Enhancements
- ▶ Subroutines
- ▶ Command Enhancements
- ▶ **New API QCAVFYNM**
- ▶ Proxy Command
- ▶ Command Documentation
- ▶ Future CL Enhancements



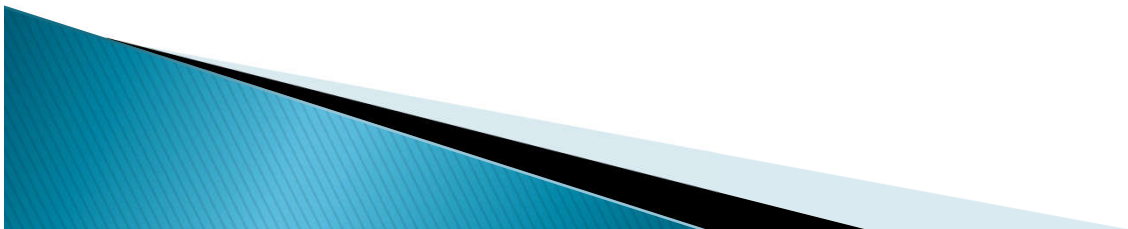
QCAVFYNM API – IBM i 6.1

- ▶ Verify Name.
 - This API verifies an input value to determine if it is a valid system name. (CPF019D means, NO!)
- ▶ Parms are:
 - CHAR(*)Data
 - CHAR(8) Format of data 'VFYN0100'
 - CHAR(*)Error.
- ▶ VFYN0100 contains (not a complete list)
 - CCSID
 - Case indicator (0=do not monospace, 1=monospace first)
 - Name type (*NAME *SNAME *CNAME)
 - Name to be verified.



Agenda:

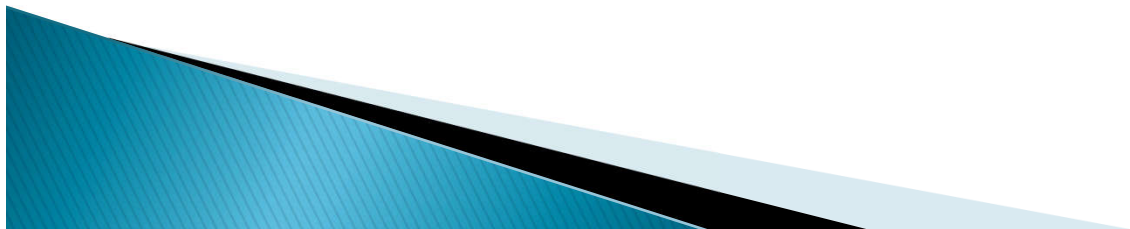
- ▶ Variable Types
- ▶ Parameter enhancements
- ▶ Multiple File Support
- ▶ Declare Processing Options
- ▶ Source member Include
- ▶ Control Flow Enhancements
- ▶ Subroutines
- ▶ Command Enhancements
- ▶ New API QCAVFYNM
- ▶ Proxy Command
- ▶ Command Documentation
- ▶ Future CL Enhancements



Proxy command support – V5R4

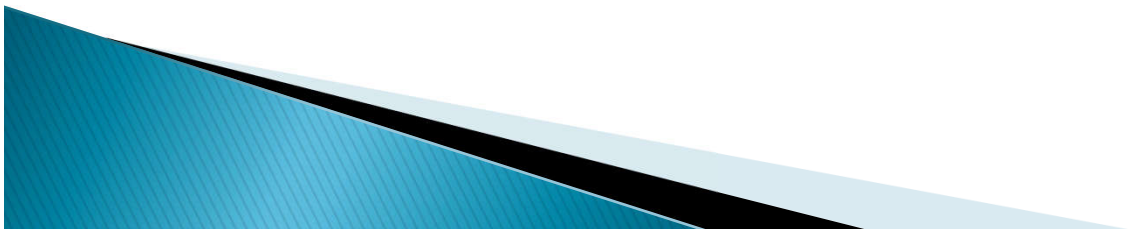
- ▶ Create a command in one library that references a command in another library
 - Proxy command has no parms it's just a pointer: 'He's over there→'
- ▶ CRTPRXCMD, CHGPRXCMD used to create and change them i.e.

```
CRTPRXCMD CMD(QGPL/SOMECMD)
              TGTCMD(MYLIBRARY/MYCMD)
              REPLACE(*NO)
```
- ▶ Proxy commands can be chained 5 levels
- ▶ Use of CHGCMD or CHGCMDDDFT operates on the **end target command** not the proxy.
 - YOU HAVE BEEN WARNED. 😊



Agenda:

- ▶ Variable Types
- ▶ Parameter enhancements
- ▶ Multiple File Support
- ▶ Declare Processing Options
- ▶ Source member Include
- ▶ Control Flow Enhancements
- ▶ Subroutines
- ▶ Command Enhancements
- ▶ New API QCAVFYNM
- ▶ Proxy Command
- ▶ Command Documentation
- ▶ Future CL Enhancements



Generate Command Documentation – V5R3

New GENCMDDOC command

▶ Run it Twice

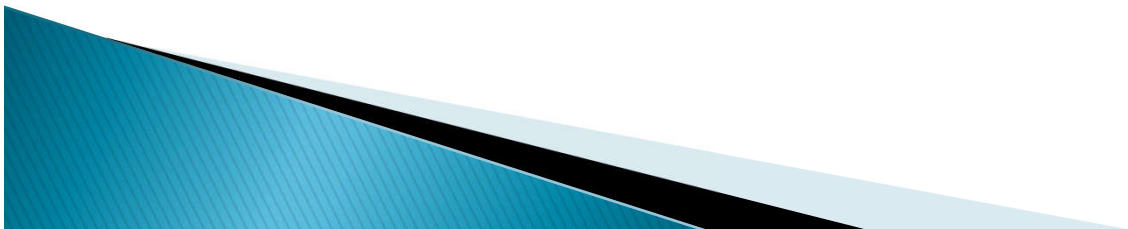
- First create a shell PNLGRP source with:
GENCMDDOC CMD(YOURLIB/YOURCMD) GENOPT(*UIM)

- You must complete the generated PNLGRP with text
- Create the PNLGRP and assign to the command
- Rerun GENCMDDOC to make nice with the html

- Second run create HTML documentation for the command

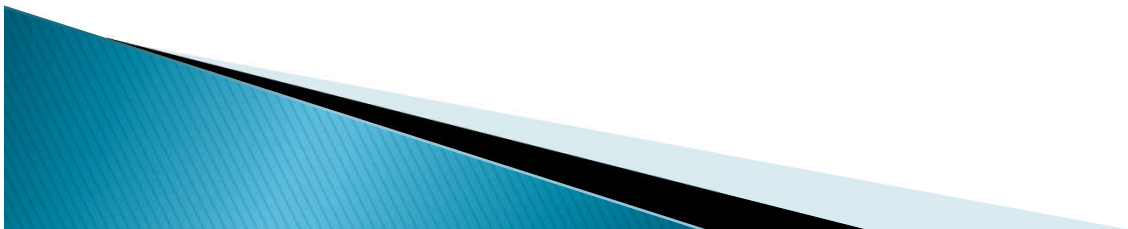
GENCMDDOC CMD(YOURLIB/YOURCMD) GENOPT(*HTML)

- Uses the command object (not source)
- Adds any UIM help (PNLGRP) text to the HTML



Agenda:

- ▶ Variable Types
- ▶ Parameter enhancements
- ▶ Multiple File Support
- ▶ Declare Processing Options
- ▶ Source member Include
- ▶ Control Flow Enhancements
- ▶ Subroutines
- ▶ Command Enhancements
- ▶ New API QCAVFYNM
- ▶ Proxy Command
- ▶ Command Documentation
- ▶ Future CL Enhancements



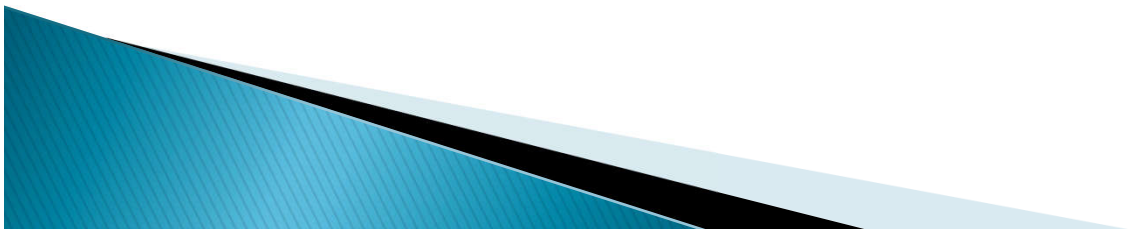
Follow-on CL Compiler Improvements

- ▶ Enhance CVTDAT to support larger year range
 - Current range is 1928 to 2071 (i 7.4)
- ▶ Compiler option to keep unreferenced CL variables
- ▶ New or extended data types for CL variables
 - *CHAR variables with LEN up to 16MB
 - *DEC variables with LEN up to 31 digits
- ▶ Single-dimension arrays and array notation syntax
- ▶ Support variable-length parameter list on PGM
- ▶ Support 31-character CL variable names
 - (Wanted by COBOL programmers 😊)



Follow-on CL Compiler Improvements

- ▶ Support structures and structure field reference notation
- ▶ Support RTNVAL parm on PGM command (ILE)
- ▶ Support “soft remove” of obsolete *CMD parameters
- ▶ Increase MAX limit on PARM and ELEM
- ▶ Support conditional prompting for *PMTRQS parms
- ▶ Allow more types of command processing code:
 - ILE procedure in a service program
 - Java method in a .jar or .zip stream file
- ▶ Support *PTR for TYPE on PARM statement
- ▶ SQL pre-compiler



Follow-on CL Compiler Improvements

- ▶ Ship CL header includes in QSYSINC library
- ▶ Increase maximum length of a CL command string
- ▶ GENCLSRC command (like GENCSRC)
 - Generate CL for record format without DCLF overhead
- ▶ Generate command processing program from *CMD
- ▶ Relax command change exit program restrictions
- ▶ Support longer object name syntax (OPM and ILE)

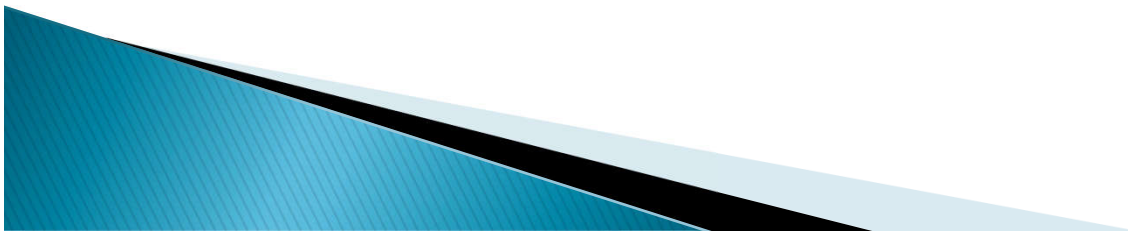


Continuing to deliver improvements

▶ Listen to customers!

Rochester wants to deliver enhancements that will delight IBM i customers, including business partners

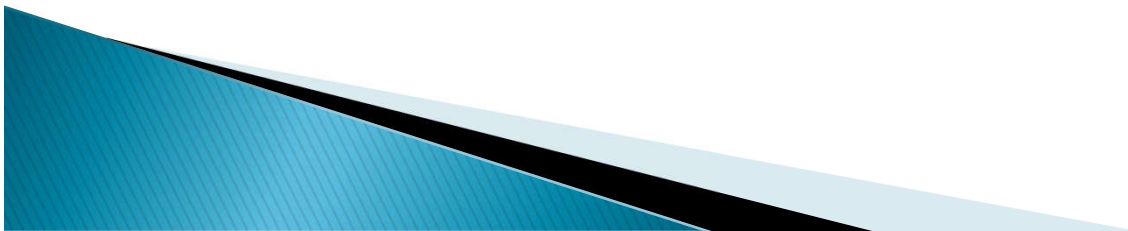
- If They're hitting the mark, tell an IBM exec!



Resources

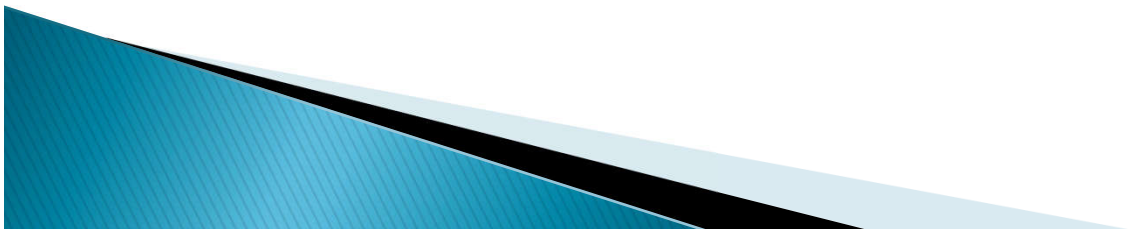
- ▶ Control Language in Knowledge Center:

www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_74/rbam6/clpro.htm



Key Points to Take Home

- ▶ The CL Language has moved forward SIGNIFICANTLY since 2004!
 - Your coding should too.
- ▶ Use of the new Control flow enhancements enables MUCH more readable code and enables the abolishment of the dreaded GOTO!
- ▶ Source includes should help with standardizing code and centralizing maintenance.
- ▶ Source in Stream Files enables use of repositories such as GIT
- ▶ Pointers combined with offsets and based variables can greatly simplify processing of users spaces returned by APIs.
- ▶ Subroutines can greatly reduce the incidence of duplicated code and improve reliability and maintainability.
- ▶ Many things cannot be done in CL and require CLLE (ILE) yet nothing in CL cannot be done in CLLE.
 - MOVE!



ENDPGM – Questions?



How to contact me:
Larry D Bolhuis
Frankeni Technology Consulting, LLC.
lbolhuis@frankeni.com
www.frankeni.com



Don't Forget to Fill Out Your Session Surveys!

1. Log on to Sched and go to your schedule
2. Click on the this session

My Schedule

Monday, May 8 • 2:00pm - 3:15pm

25AG CL Enhancements V5R3 to i 7.3 and Beyond



3. Click on the feedback survey link above the session abstract

<http://sched.co/7hTQ>

 Tweet

 Share

Feedback Survey



4. To fill out additional surveys go back to your schedule and click on the next session