

z/OS  
2.5

*JES2 Macros*



**Note**

Before using this information and the product it supports, read the information in [“Notices” on page 501.](#)

This edition applies to Version 2 Release 5 of z/OS® (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2022-01-26

© **Copyright International Business Machines Corporation 1988, 2022.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Figures.....</b>	<b>xix</b>
<b>Tables.....</b>	<b>xxi</b>
<b>About this document.....</b>	<b>xxiii</b>
Who should use this document.....	xxiii
Where to find more information.....	xxiii
<b>How to send your comments to IBM.....</b>	<b>xxv</b>
If you have a technical problem.....	xxv
<b>Summary of changes.....</b>	<b>xxvii</b>
Summary of changes for z/OS JES2 Macros for Version 2 Release 5 (V2R5).....	xxvii
Summary of changes for z/OS Version 2 Release 4 (V2R4).....	xxvii
Summary of changes for z/OS Version 2 Release 3 (V2R3).....	xxviii
<b>Chapter 1. Macro overview.....</b>	<b>1</b>
How to read syntax diagrams.....	2
Symbols.....	2
Syntax items.....	2
Syntax examples.....	3
Macro expansion.....	4
Specify JES2 macro instructions.....	4
Basic notation used to describe macro instructions.....	5
Operand representation.....	5
Operands with value mnemonics.....	6
Coded value operands.....	7
Metasymbols.....	7
Special register notation.....	8
Register stability.....	8
Macro selection table.....	8
Using the \$JCTX macro extension service.....	10
Determining the amount of SPOOL space used by SPOOLed \$JCT extensions .....	10
Using local \$JCT extensions.....	11
Examples of the \$JCTX macro extension service.....	11
Example 1: Transmitting separator notes through \$JCT extensions.....	11
Example 2: Using \$JCTX extensions to maintain spool compatibility.....	12
Sample Exit 6 for spool compatibility.....	12
<b>Chapter 2. JES2 programmer macros.....</b>	<b>15</b>
\$\$POST – Post a JES2 event complete from another task.....	15
Format description.....	15
Environment.....	18
\$\$WTO – JES2 subtask write to operator.....	18
Format description.....	19
Environment.....	20
\$\$WTOR – JES2 subtask write to operator with reply.....	20
Format description.....	20
Environment.....	20

\$#ADD – Add a work/characteristics JOE pair to the JOT.....	20
Format description.....	21
Return codes.....	22
Environment.....	22
\$#ALCHK – Obtain a spool record for output checkpointing.....	22
Format description.....	22
Environment.....	23
\$#BLD – Format JOEs.....	23
Format description.....	23
Environment.....	24
\$#BUSY – Set or test the busy system indicator of a JOE.....	24
Format description.....	24
Environment.....	26
\$#CAN – Cancel all work items not currently being processed for a specific job.....	26
Format description.....	27
Environment.....	27
\$#CHK – Process print/punch checkpoint spool I/O.....	27
Format description.....	27
Environment.....	28
\$#DISPRO – Process JOE disposition.....	28
Format description.....	28
Environment.....	29
\$#GET – Search the JOT class queues for an output element which matches the requesting specification.....	29
Format description.....	30
Environment.....	31
\$#GETHDJ – Get held JOE.....	31
Format description.....	31
Environment.....	32
\$#JOE – Find and validate queue.....	32
Format description.....	33
Environment.....	35
\$#JWEL – JOE writer exclude list (JWEL) services.....	35
Format description.....	36
Programming considerations.....	39
Return codes.....	39
Environment.....	40
\$#MOD – Move a work JOE from one queue to another in the JOT.....	40
Format description.....	40
Environment.....	40
\$#POST – Post output device processors.....	40
Format description.....	41
Environment.....	41
\$#PUT – Return an unfinished job output element (JOE) to the JOT for later processing.....	41
Format description.....	42
Environment.....	42
\$#REM – Remove a work/characteristics JOE pair from the JOT.....	42
Format description.....	43
Environment.....	43
\$#REP – Replace a work or characteristics JOE.....	44
Format description.....	44
Return codes.....	44
Environment.....	45
\$#TJEV – Manage the thread JOE exclusion vector.....	45
Format description.....	45
Return codes.....	46
Environment.....	46
\$ACTIVE – Specify processor is active.....	46

Format description.....	47
Environment.....	47
<b>\$ALLOC – Allocate a unit record device.....</b>	<b>47</b>
Format description.....	47
Environment.....	47
<b>\$ALESERV – JES2 ALET services.....</b>	<b>47</b>
Format description.....	48
Programming notes.....	48
Return codes.....	49
Environment.....	49
<b>\$AMODE – Set the addressing mode.....</b>	<b>49</b>
Format description.....	49
Environment.....	50
<b>\$ARMODE – JES2 multi-address space access.....</b>	<b>50</b>
Format description.....	50
Environment.....	51
<b>\$BERTTAB – Map block extension reuse table (BERT) table entries.....</b>	<b>51</b>
Format description.....	51
Environment.....	53
<b>\$BFRBLD – Construct a JES2 buffer prefix.....</b>	<b>53</b>
Format description.....	53
Environment.....	53
<b>\$BLDMSG – Build a message line.....</b>	<b>53</b>
Format description.....	55
Register contents when \$BLDMSG is invoked.....	60
Register contents on exit from \$BLDMSG.....	61
Return codes.....	61
Usage notes.....	62
Environment.....	62
<b>\$BLDQC – Call the quick cell build/extend routine.....</b>	<b>62</b>
Format description.....	62
Environment.....	63
<b>\$BLDTGB – Queue TGBs to the HASPOOL processor.....</b>	<b>63</b>
Format description.....	63
Environment.....	63
<b>\$BPXCALL – Call omvs services.....</b>	<b>63</b>
Format description.....	63
Environment.....	64
<b>\$CALL – Call a subroutine from JES2.....</b>	<b>64</b>
Format description.....	65
Programming considerations.....	67
Environment.....	67
<b>\$CBIO – Control block I/O routine.....</b>	<b>67</b>
Format description.....	68
Register contents when \$CBIO is invoked.....	72
Register contents on exit from \$CBIO.....	72
Return codes.....	73
Environment.....	73
<b>\$CFSEL – Select label to process a command operand string.....</b>	<b>73</b>
Format description.....	74
Register contents when \$CFSEL is invoked.....	76
Register contents on exit from \$CFSEL.....	76
Return codes.....	76
Usage notes.....	76
Environment.....	77
Examples.....	77
Example 1.....	77
Example 2.....	77

Example 3.....	78
Example 4.....	78
Example 5.....	78
Example 6.....	78
Example 7.....	79
\$CHECK – Check checkpoint write completion.....	79
Format description.....	80
Environment.....	80
\$CKPT – Schedule an element checkpoint.....	81
Format description.....	81
Environment.....	82
\$CPOOL – Build, delete, modify, or query a cell pool.....	82
Format description - Execute form.....	83
Format description - List form.....	84
Return codes.....	88
Environment.....	89
\$CWTO – Command processor write to operator.....	89
Format description.....	90
Usage notes.....	91
Environment.....	91
\$DCBDYN – Call the dynamic DCB service routine.....	92
Format description.....	92
Return codes.....	92
Environment.....	92
\$DCTDYN – Call the dynamic DCT service routine.....	92
Format description.....	92
Return codes.....	93
Environment.....	93
\$DCTTAB – Map DCT table entries.....	93
Format description.....	94
Environment.....	100
\$DEST – Convert symbolic destinations and binary route codes.....	100
Format description.....	101
Return codes.....	104
Environment.....	104
\$DESTDYN – Attach a JES2 DESTID.....	104
Format description.....	105
Return codes.....	105
Environment.....	106
\$DILBERT – Do it later BERT services.....	106
Format description.....	107
Environment.....	110
Return codes.....	110
\$DISTERR – Indicate disastrous error.....	110
Format description.....	111
Environment.....	112
\$DOGBERT – Deliver or get BERT data.....	112
Format description.....	113
Environment.....	118
Return codes.....	118
\$DOGCAT – Deliver or get CAT (class attribute table).....	119
Format description.....	120
Environment.....	122
Return codes.....	122
\$DOGDJB – Deliver or Get DJB.....	123
Format description.....	124
Environment.....	125
Registers on entry.....	126

Registers on exit.....	126
Return codes.....	126
\$DOGJOE – Deliver or get JOE.....	126
Format description.....	127
Return codes.....	132
Environment.....	132
\$DOGJQE – Deliver or get JQE.....	132
Format description.....	133
Return codes.....	137
Environment.....	137
\$DOGWSCQ – Deliver or get workload management (WLM) service class.....	137
Format description.....	138
Environment.....	140
Return codes.....	140
\$DOM – Delete operator message.....	140
Format description.....	140
Environment.....	140
\$DORMANT – Specify processor is inactive.....	141
Format description.....	141
Environment.....	141
\$DSERV – Obtain or free a DSERV pointer.....	141
Format description.....	142
Return codes.....	142
Environment.....	143
\$DSPSERV – JES2 data space services.....	143
Format description.....	144
Programming considerations.....	147
Environment.....	147
\$DTEDYN – Call the dynamic DTE service routines.....	147
Format description.....	147
Environment.....	149
\$DTETAB – Build and map the DTE tables.....	149
Format description.....	149
Environment.....	152
\$DVIDBLD – Build a device name from a device identifier.....	152
Format description.....	153
Environment.....	153
\$ENTRY – Provide entry to JES2 assembly module.....	153
Format description.....	154
Environment.....	156
\$ENVIRON – Set assembly environment.....	156
Format description.....	157
Environment.....	158
\$ERROR – Indicate catastrophic error.....	158
Format description.....	159
Environment.....	161
\$ESTAE – JES2 error recovery environment.....	161
Format description.....	161
Environment.....	163
\$EXCP – Execute JES2 channel program.....	163
Format description.....	163
Environment.....	164
\$EXIT – Provide exit point.....	164
Format description.....	164
Environment.....	166
\$EXTP – Initiate remote terminal input/output operation.....	166
Format description.....	166
Environment.....	167

\$FIFOBLK – Manage blocking of a FIFO queue.....	167
Format description.....	167
Environment.....	167
Registers on entry.....	168
Registers on exit.....	168
Return codes.....	168
\$FIFODEQ – Remove an element from a FIFO queue.....	168
Format description.....	168
Environment.....	169
\$FIFOENQ – Add an element to a FIFO queue.....	169
Format description.....	169
Environment.....	170
\$FIFOQTQ - Dequeue an entire FIFO queue.....	170
Format description.....	170
Environment.....	171
\$FRECEL – Free an extended common storage area (ECSA) cell.....	171
Format description.....	171
Environment.....	171
\$FRECMB – Free a console message buffer.....	172
Format description.....	172
Environment.....	172
\$FREEBUF – Return a JES2 buffer to the JES2 buffer pool.....	172
Format description.....	172
Environment.....	173
\$FREJLOK – Release the JES2 job lock.....	173
Format description.....	173
Environment.....	174
\$FRELOK – Free the MVS CMS lock, LOCAL, or JES2 job lock.....	175
Format description.....	175
Environment.....	176
\$FREMLOK – Release the MVS CMS or LOCAL job lock.....	176
Format description.....	176
Environment.....	177
\$FREMAIN – Branch-entry FREEMAIN services.....	177
Format description.....	178
Environment.....	179
\$FREQC – Free quick cell.....	180
Format description.....	180
Environment.....	180
\$FRETBUF – Free TCP buffer.....	181
Format description.....	181
Environment.....	181
Return codes.....	181
\$FREUCBS – Free UCB parameter list storage.....	181
Format description.....	181
Environment.....	182
\$FREUNIT – Release a unit device control table (DCT).....	182
Format description.....	182
Environment.....	182
\$FSILINK – Link the functional subsystem interface.....	182
Format description.....	182
Environment.....	183
\$GETABLE – Get HASP/user table entries.....	183
Format description.....	183
Environment.....	184
\$GETADDR – Get a control block address.....	184
Format description.....	185
Environment.....	187



\$GETASCB – Retrieve the primary, secondary, or home ASCB.....	187
Format description.....	187
Environment.....	188
\$GETBLK – Get a storage cell from a free cell pool.....	188
Format description.....	188
Environment.....	188
\$GETBUF – Acquire a buffer from a JES2 buffer pool.....	189
Format description.....	189
Return codes.....	191
Environment.....	191
\$GETCEL – Acquire an extended common storage (ECSA) area cell.....	191
Format description.....	192
Return codes.....	192
Environment.....	193
\$GETCMB – Get console message buffers.....	193
Format description.....	193
Return codes.....	194
Register contents when \$GETCMB returns control.....	194
Environment.....	194
\$GETHP – Get high private cell pool.....	194
Format description.....	195
Environment.....	195
Programming requirements.....	195
\$GETJLOK – Acquire the JES2 job lock.....	195
Format description.....	196
Environment.....	197
\$GETLOK – Acquire the MVS CMS, LOCAL, or JES2 job lock.....	197
Format description.....	197
Environment.....	198
\$GETMLOK – Acquire the MVS CMS or LOCAL job lock.....	198
Format description.....	198
Environment.....	199
\$GETMAIN – Branch-entry GETMAIN services.....	199
Format Description.....	200
Environment.....	203
\$GETQC – Call the quick cell get routine.....	203
Format description.....	203
Environment.....	204
\$GETRTN – Get the address of a routine.....	204
Format description.....	204
Environment.....	205
\$GETSMFB – Acquire a JES2 SMF buffer from the JES2 SMF buffer pool.....	205
Format description.....	206
Environment.....	206
\$GETTBUF – Get TCP buffer.....	206
Format description.....	206
Environment.....	207
Return codes.....	207
\$GETUCBS – Obtain a UCB address.....	207
Format description.....	207
Return codes.....	207
Programming considerations.....	208
Environment.....	208
\$GETUNIT – Acquire a unit device control table (DCT).....	208
Format description.....	208
Environment.....	209
\$GETWORK – Obtain a work area.....	209
Format description.....	209

Environment.....	210
\$IOERROR – Log input/output error.....	210
Format description.....	210
Environment.....	210
\$IOTBLD – Build an input/output table (IOT).....	210
Format Description.....	211
Return codes.....	211
Environment.....	212
\$JBIDBLD – Build a JES2 job ID from a binary job number.....	212
Format description.....	212
Environment.....	213
\$JCAN – Cancel job.....	213
Format description.....	213
Environment.....	215
\$JCORBLD – Build a job correlator.....	215
Format description.....	215
Environment.....	215
\$JCTXADD – Add a \$JCT control block extension.....	215
Format description.....	216
Return codes.....	218
Environment.....	218
Programming requirements.....	218
Restrictions.....	219
Registers on entry.....	219
Registers on exit.....	219
Example.....	219
\$JCTXEXP – Expand a \$JCT Control block extension.....	220
Format description.....	220
Return codes.....	221
Environment.....	222
Programming requirements.....	222
Restrictions.....	222
Registers on entry.....	222
Registers on exit.....	223
Example.....	223
\$JCTXGET – Get a \$JCT extension.....	223
Format description.....	224
Return codes.....	225
Environment.....	225
Programming requirements.....	225
Restrictions.....	226
Registers on entry.....	226
Registers on exit.....	226
Example.....	226
\$JCTXREM – Remove a \$JCT control block extension.....	226
Format Description.....	227
Return codes.....	227
Environment.....	228
Programming requirements.....	228
Restrictions.....	228
Registers on entry.....	228
Registers on exit.....	229
Example.....	229
\$JQEJNUM – Obtain JQE job number.....	229
Format description.....	229
Return codes.....	229
Environment.....	229
\$JQESERV – User environment JQE services.....	230

Format description.....	230
Environment.....	233
Registers on entry.....	233
Registers on exit.....	233
Return codes.....	234
\$LOGMSG – Log a job-related message.....	234
Format description.....	234
Return codes.....	235
Environment.....	235
\$MID – Assign JES2 message identification.....	236
Format description.....	236
Environment.....	236
\$MODCHK – Load module verification.....	236
Format description.....	237
Return codes.....	239
Environment.....	239
\$MODELET – Delete a load module .....	240
Format description.....	240
Return codes.....	240
Environment.....	240
\$MODEND – Generate end of module.....	241
Format description.....	241
Environment.....	241
\$MODLOAD – Load module load.....	241
Format description.....	241
Return codes.....	242
Environment.....	243
\$MODULE – Prepare a JES2 module or expand control block mappings.....	244
Preparing a JES2 module.....	244
Expanding MVS or JES2 control block mappings.....	244
Format description - Preparing a JES2 module.....	245
Format description - Expanding control block mappings.....	245
Parameter descriptions.....	246
Environment.....	262
\$MSG – Write to operator message area.....	262
Format description.....	262
Environment.....	263
\$MVCL – Move more than 256 bytes of storage.....	263
Format description.....	263
Environment.....	264
\$NATGET – Locate a NAT element .....	264
Format description.....	264
Return codes.....	265
Environment.....	266
\$NHDADD – Adds an installation-defined section to an NJE data area.....	266
Format description.....	266
Return codes.....	267
\$NHDEXP – Expand an NJE data area.....	268
Format description.....	268
Return codes.....	269
\$NHGET – Get the network header section.....	269
Format description.....	269
Environment.....	270
\$NHREM – Removes an installation-defined section from an NJE data area.....	270
Format description.....	271
Return codes.....	271
\$NHDXMT – Transmitting an NJE data area across the network.....	272
Format description.....	272

Return codes.....	273
Environment.....	273
\$NITSYNC – Synchronize NIT settings.....	273
Format description.....	274
Return codes.....	274
Environment.....	275
\$NJETRC - NJE subdevice rolling trace.....	275
Format description.....	275
\$NOTIFY – Send a notify message to a specific user ID and node.....	275
Format description.....	276
Return codes.....	277
Environment.....	277
\$PAIR – Define a table pair.....	277
Format description.....	277
Environment.....	278
\$PATCHSP – Generate patch space.....	278
Format description.....	278
Environment.....	279
\$PBLOCK – Block letter services.....	279
Format description.....	279
Environment.....	280
\$PCEDYN – Attach or delete a JES2 PCE.....	280
Format description.....	280
Environment.....	281
\$PCETAB – Generate or map PCE table entries.....	281
Format description.....	282
Environment.....	287
\$PCETERM – Terminate a processor control element (PCE) .....	287
Format description.....	287
Environment.....	288
\$PDBBLD – Build a peripheral data definition block (PDDB).....	288
Format description.....	288
Return codes.....	288
Environment.....	289
\$PDBFIND – Locate a peripheral data definition block (PDDB).....	289
Format description.....	289
Return codes.....	289
Environment.....	289
\$PGSRVC – Perform a virtual page service.....	290
Format description.....	290
Environment.....	291
\$POST – Post a JES2 event complete.....	291
Format description.....	291
Environment.....	294
\$POSTQ – Quick post facility.....	295
Format description.....	295
Environment.....	295
\$POSTXEQ – Wake up the EXECUTION PCE.....	295
Format description.....	295
Environment and Serialization.....	296
\$PRPUT – Create separator pages.....	296
Format description.....	296
Return codes.....	297
Environment.....	297
\$PURGE – Return direct-access space.....	297
Format description.....	297
Environment.....	297
\$PUTABLE – Add HASP/user table entry.....	297

Format description.....	298
Return codes.....	298
Environment.....	299
\$QADD – Add job queue element to the JES2 job queue.....	299
Format description.....	299
Environment.....	299
\$QBUSY – Set or test the JQE busy–system indicator.....	300
Format description.....	300
Environment.....	301
\$QCTGEN – Define a quick cell control table.....	302
Format description.....	302
Environment.....	302
\$QGET – Obtain a job queue element from the JES2 job queue.....	302
Format description.....	303
Environment.....	304
\$QJIX – JES2 job number services.....	304
Format description.....	305
Return codes.....	305
Environment.....	306
\$QJQE – Obtain address of JQE queue head.....	306
Format description.....	307
Environment.....	309
\$QLOC – Locate job queue element (JQE) for specific job.....	309
Format description.....	309
Return codes.....	310
Environment.....	310
\$QLOCNXT– Find next job number after current JQE in JIX.....	310
Format description.....	310
Return codes.....	311
Environment.....	311
\$QMOD – Modify job queue element in JES2 job queue.....	311
Format description.....	312
Environment.....	313
\$QPUT – Return job queue element to the JES2 job queue.....	313
Format description.....	313
Environment.....	314
\$QREM – Remove job queue element from JES2 job queue.....	314
Format description.....	314
Environment.....	314
\$QSUSE – Synchronize to use shared queues.....	314
Format description.....	315
Environment.....	315
\$QUESMFB – Queue a JES2 SMF buffer on the busy queue.....	316
Format description.....	316
Environment.....	316
\$QUEUE – Maintain a first–in–first–out (FIFO) queue.....	316
Format description.....	316
Environment.....	317
\$RDRTAB – Build table to redirect responses to specific commands.....	317
Format description.....	317
Environment.....	318
\$REPLYV – Generate \$REPLYV table entries.....	318
Format description.....	318
Environment.....	319
\$RESTORE – Restore registers from the save area.....	319
Format description.....	319
Environment.....	320
\$RETABLE – Remove HASP/user table entry.....	320

Format description.....	320
Return codes.....	321
Environment.....	321
\$RETBK – Return a storage cell to a free-cell pool.....	321
Format description.....	321
Environment.....	322
\$RETSAVE – Return a JES2 save area.....	322
Format description.....	322
Environment.....	322
\$RETURN – Restore registers, free the JES2 save area, and return to the caller.....	322
Format description.....	323
Environment.....	324
\$RETWORK – Return a work area.....	324
Format description.....	324
Environment.....	324
\$RMSGQUE – Queue message to JES2.....	324
Format description.....	325
Register usage.....	326
Return codes (R15 on exit).....	326
Environment.....	326
\$RUSE – Establish USING on a register.....	326
Format description.....	326
Environment.....	327
\$SAVE – Obtain JES2 save area and save registers.....	327
Format description.....	328
Environment.....	329
\$SCAN – Scan JES2 parameter statements.....	329
Format description.....	330
Environment.....	333
Registers on entry.....	333
Registers on exit.....	333
Return codes.....	334
Other considerations.....	334
\$SCANB – Backup storage for a scan.....	334
Format description.....	335
Environment.....	335
\$SCANCOM – Call the \$SCAN facility comment service routine.....	336
Format description.....	336
Return codes.....	336
Environment.....	336
\$SCAND – Call the \$SCAN facility display service routine.....	336
Format description.....	337
Environment.....	340
\$SCANDIA – \$SCAN diagnostic message service.....	340
Format description.....	340
Environment.....	340
\$SCANTAB – Create a scan table.....	341
Format description.....	341
Environment.....	354
\$SDUMP – Take an SDUMP of storage.....	355
Format description.....	355
Environment.....	356
\$SEAS – Security authorization services.....	356
Format description.....	357
Return codes.....	360
Usage notes.....	361
Environment.....	361
\$SEPPDIR – Create a user peripheral data information record (PDIR).....	361

Format description.....	361
Environment.....	361
\$SETAFF – Set \$SIDAFF into correct affinity.....	361
Format description.....	362
Return codes.....	364
\$SETIDAW – Set indirect data access word (IDAW).....	364
Format description.....	364
Environment.....	365
\$SETRP – Set recovery processing options.....	365
Format description.....	365
Environment.....	365
\$SJBFind – Locate a subsystem job block (SJB).....	365
Format description.....	366
Return codes.....	366
Programming requirement.....	367
Environment.....	367
\$SJBLOCK – Lock a specific subsystem job block (SJB).....	367
Format description.....	367
Return codes.....	367
Environment.....	368
\$SJBREQ – Requeue a specific subsystem job block (SJB).....	368
Format description.....	368
Return codes.....	368
Environment.....	368
\$SSIBEGN – Begin a subsystem interface (SSI) function.....	369
Format description.....	369
Environment.....	370
\$SSIEND – End a subsystem interface (SSI) function.....	370
Format description.....	370
Environment.....	370
\$STCK – Call the \$STCK service routine.....	370
Format description.....	371
Environment.....	371
\$STIMER – Set interval timer.....	371
Format description.....	371
Environment.....	372
\$STMTLOG – Log an initialization statement.....	372
Format description.....	372
Environment.....	373
\$STMTTAB – Card Scan table entry.....	373
Format description.....	374
Return codes.....	377
\$STORE – Store registers in the current processor save area.....	377
Format description.....	377
Environment.....	378
\$SUBIT – Initiate subtask queueing.....	378
Format description.....	378
Return codes.....	380
Environment.....	380
\$SYMREC – Create and issue a symptom record.....	380
Format description.....	381
Return codes.....	381
Environment.....	381
\$SYMTAB – Create a symptom record table.....	381
Format description.....	382
Environment.....	385
\$TIDTAB – Generate a trace ID table DSECT.....	385
Format description.....	385

Environment.....	386
\$TOKENSR - Create a name/token pair.....	386
Format description.....	386
Return codes.....	388
\$TRACE – Trace a JES2 activity.....	388
Format description.....	389
Environment.....	391
\$TRACK – Acquire a direct-access track address.....	391
Format description.....	391
Return codes.....	392
Environment.....	392
\$TTIMER – Test interval timer.....	392
Format description.....	393
Environment.....	393
\$VERIFY – Verify a control block.....	393
Format description.....	394
Return codes.....	396
Environment.....	396
\$VERTAB – Build in-line verification tables.....	396
Format description.....	396
Environment.....	398
\$VFL – Variable field length instruction operation.....	398
Format description.....	398
Environment.....	398
\$WAIT – Wait for a JES2 event.....	399
Format description.....	399
Environment.....	402
\$WSSETUP – Set values required for work selection.....	402
Format description.....	402
Environment.....	403
\$WSTAB – Map and generate the work selection table entries.....	403
Format description.....	403
Environment.....	411
\$WTO – JES2 Write to operator – JES2 environment.....	411
Format description – Standard form.....	412
Format description – Execution form.....	412
Format description – List form.....	413
Environment.....	417
\$WTO – JES2 Write to operator – User and subtask environment.....	417
Format description – Standard form.....	418
Environment.....	420
\$XECBSRV – Interface for extended event control block (XECB) services.....	420
Format description.....	420
Environment.....	421
\$XMPOST – POST task in another address space.....	421
Format description.....	422
Environment.....	423

## **Appendix A. Using JES2 table pairs.....425**

What are JES2 table pairs?.....	425
JES2 table pairs versus JES2 exits.....	425
Concepts.....	426
Master control table.....	428
General table coding conventions.....	429
Dynamic tables versus installation tables.....	429
Examples of table pairs.....	430
Processor control elements (PCE) tables.....	430



PCE control blocks and macros.....	430
A JES2 PCE table.....	432
An installation PCE table.....	433
A dynamic PCE table.....	433
Coding the other pieces.....	435
Daughter task element (DTE) tables.....	436
DTE control blocks and macros.....	436
A JES2 DTE table.....	437
An installation DTE table.....	438
A dynamic DTE table.....	438
Coding the other pieces.....	439
Work selection (WS) tables.....	439
WS control blocks and macros.....	439
A JES2 WS table.....	440
An installation WS table.....	440
A dynamic WS table.....	440
Coding the other pieces.....	442
Trace identifiers (TID) tables.....	442
TID control blocks and macros.....	442
A JES2 TID table.....	443
An installation TID table.....	443
A dynamic TID table.....	443
Coding the other pieces.....	444
Creating a trace table using the \$TRACE macro.....	445
Block extension reuse table (BERT) tables.....	445
BERT control blocks and macros.....	445
A JES2 BERT table.....	446
An installation BERT table.....	446
A dynamic BERT table.....	446
Coding the other pieces.....	447
JES2 \$SCAN facility.....	447
\$SCAN-related control blocks.....	448
Implementing \$SCAN tables.....	448
Examples of \$SCAN tables.....	455

## **Appendix B. Table pairs coding example.....459**

\$USERCBS - Generates user control blocks.....	459
\$SCYWORK - Processor work area.....	460
\$SCDWORK - Subtask work area.....	460
\$UCT - User communication table.....	461
EXIT 0 - Initialization.....	462
User extension code and tables.....	466
USCTPCE - INITIAL ENTRY POINT.....	468
USCTDTE - SECURITY SUBTASK, INITIAL ENTRY POINT.....	470
USCTDTE - SECURITY SUBTASK, MAIN PROCESSING.....	472
USCTDTE - SECURITY SUBTASK, TERMINATION.....	472
TROUTE255 - TRACING ROUTINE FOR SAF CALL.....	473
WSTRKGRP - WORK SELECTION ROUTINE.....	474
TABLES.....	475

## **Appendix C. Miscellaneous facilities support.....479**

Generalized JES2 dispatcher support.....	479
Data space usage.....	479
\$ARMODE.....	479
\$DSPSERV.....	479
General purpose subtasking facility.....	480
Using the general purpose subtasking facility.....	480

<b>Appendix D. Accessing checkpoint control blocks outside the JES2 main task.....</b>	<b>481</b>
Typical macro calls.....	481
\$QJQE.....	481
\$#JOE .....	482
\$DOGBERT .....	482
\$DOGCAT .....	483
\$DOGJQE .....	483
\$DOGWSCQ.....	484
\$QLOC.....	484
<b>Appendix E. Invoking the security authorization facility (SAF).....</b>	<b>487</b>
Using \$SEAS to invoke SAF.....	487
<b>Appendix F. Techniques for writing multi-environment access.....</b>	<b>491</b>
<b>Appendix G. Accessibility.....</b>	<b>497</b>
Accessibility features.....	497
Consult assistive technologies.....	497
Keyboard navigation of the user interface.....	497
Dotted decimal syntax diagrams.....	497
<b>Notices.....</b>	<b>501</b>
Terms and conditions for product documentation.....	502
IBM Online Privacy Statement.....	503
Policy for unsupported hardware.....	503
Minimum supported hardware.....	503
Programming Interface Information.....	504
Trademarks.....	504
<b>Index.....</b>	<b>505</b>

---

# Figures

1. Determine the Amount of Spool Space Used by \$JCTX Extensions.....	11
2. Table Pairs: A Diagrammatic View.....	426
3. Common PCE Area Structure.....	431
4. PCE Tables - Save Area Chaining.....	432
5. The JES2 PCE Table.....	432
6. Example of an Installation PCE Table.....	433
7. Example of a Dynamic PCE Table.....	433
8. The JES2 DTE Table.....	437
9. Example of An Installation DTE Table.....	438
10. Example of a Dynamic DTE Table.....	438
11. The JES2 WS Table.....	440
12. Example of an Installation WS Table.....	440
13. Example of a Dynamic WS Table.....	441
14. Trace Table Structure.....	443
15. The JES2 TID Table.....	443
16. Example of an Installation TID Table.....	443
17. Example of a Dynamic TID Table.....	444
18. The JES2 BERT Table.....	446
19. Example of an Installation BERT Table.....	446
20. Example of a Dynamic BERT Table.....	446
21. Three Examples of \$SCANTAB Tables.....	456
22. Typical \$DSERV call when obtaining a JQE with \$QJQE.....	482
23. Typical \$DSERV call when validating a queue with \$#JOE.....	482

24. Typical \$DSERV call when obtaining a BERT data with \$DOGBERT.....	483
25. Typical \$DSERV call when obtaining a CAT with \$DOGCAT.....	483
26. Typical \$DSERV call when obtaining an artificial JQE (JQA) with \$DOGJQE.....	484
27. Typical \$DSERV call when obtaining a workload management service class with \$DOGWSCQ.....	484
28. Typical \$DSERV call when obtaining a job queue element with \$QLOC.....	485

---

# Tables

1. Syntax examples.....	3
2. JES2 Macro Selection Table.....	9
3. Summary of \$#JWEL Parameter Requirements and Restrictions.....	39
4. \$DOGBERT Parameter Table (1 of 2).....	117
5. \$DOGBERT Parameter Table (2 of 2).....	117
6. \$JQESERV Parameter Table.....	232
7. MVS DSECTIDs That Can Be Specified on \$MODULE.....	250
8. JES2 DSECTIDs That Can Be Specified on \$MODULE.....	251
9. \$QJQE Standard Queue Head Names.....	307
10. JES2 Reserved Master Control Table Names.....	450



## About this document

---

This document supports z/OS (5650-ZOS).

This document provides information about the use and syntax of JES2 executable macro instructions.

## Who should use this document

---

This document is intended for JES2 system programmers or for anyone responsible for customizing JES2.

## Where to find more information

---

Where necessary, this document references information in other documents, using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see [\*z/OS Information Roadmap\*](#).





# How to send your comments to IBM

---

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

**Important:** If your comment regards a technical question or problem, see instead [“If you have a technical problem”](#) on page xxv.

Submit your feedback by using the appropriate method for your type of comment or question:

## Feedback on z/OS function

If your comment or question is about z/OS itself, submit a request through the [IBM RFE Community](#) ([www.ibm.com/developerworks/rfe/](http://www.ibm.com/developerworks/rfe/)).

## Feedback on IBM® Documentation function

If your comment or question is about the IBM Documentation functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Documentation Support at [ibmdocs@us.ibm.com](mailto:ibmdocs@us.ibm.com).

## Feedback on the z/OS product documentation and content

If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to [mhvrcfs@us.ibm.com](mailto:mhvrcfs@us.ibm.com). We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS JES2 Macros, SA32-0996-50
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive authority to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

# If you have a technical problem

---

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the [IBM Support Portal](#) ([support.ibm.com](http://support.ibm.com)).
- Contact your IBM service representative.
- Call IBM technical support.



## Summary of changes

---

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

**Note:** IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under [IBM z/OS Product Documentation Update Policy \(www-01.ibm.com/servers/resourcelink/svc00100.nsf/pages/ibm-zos-doc-update-policy?OpenDocument\)](http://www-01.ibm.com/servers/resourcelink/svc00100.nsf/pages/ibm-zos-doc-update-policy?OpenDocument).

## Summary of changes for z/OS JES2 Macros for Version 2 Release 5 (V2R5)

---

The following content is new, changed, or no longer included in V2R5.

### New

The following content is new.

#### January 2022 refresh

- APAR OA61231 updated “[\\$MODULE – Prepare a JES2 module or expand control block mappings](#)” on page 244 to add MVS DSECTIDs:  
BPXYIOCC  
EZAENF80
- APAR OA61231 updated “[\\$MODULE – Prepare a JES2 module or expand control block mappings](#)” on page 244 to add JES2 DSECTIDs:  
\$URIMAP
- APAR OA61231 updated “[Implementing \\$SCAN tables](#)” on page 448 to add \$SCAN Targets:  
EDS  
ESQ

### Changed

The following content is changed.

- None

### Deleted

The following content was deleted.

- None

## Summary of changes for z/OS Version 2 Release 4 (V2R4)

---

The following changes are made for z/OS Version 2 Release 4 (V2R4).

### New

#### August 2020 refresh

- “[\\$MODULE – Prepare a JES2 module or expand control block mappings](#)” on page 244 updated to add MVS DSECTIDs:

DSAB  
ITRLP  
ITSPP  
QMPA

- “[\\$MODULE – Prepare a JES2 module or expand control block mappings](#)” on page 244 updated to add JES2 DSECTIDs:

\$CDI  
\$CKPINFO  
\$POLICY

#### **Prior or June 2020 refresh**

- “[\\$FREMAIN – Branch-entry FREEMAIN services](#)” on page 177 and “[\\$GETMAIN – Branch-entry GETMAIN services](#)” on page 199 are updated to add the EXECUTABLE keyword.
- “[\\$MODULE – Prepare a JES2 module or expand control block mappings](#)” on page 244, [Table 7](#) on page 250 is updated to add DSECTIDs:

CKPD  
HWTJ  
LWA  
RAX  
TRKCALC

“[\\$MODULE – Prepare a JES2 module or expand control block mappings](#)” on page 244, [Table 8](#) on page 251 is updated to add DSECTIDs:

\$ARTABL  
\$DTECKDA  
\$DTELIM  
\$LIMITS  
\$PCYINTR  
\$PCYPARS  
\$PCYWORK  
\$PRA  
\$XACTTAB  
\$XFATAB  
\$XOPTAB  
\$XVCTAB

#### **Changed**

None.

## **Summary of changes for z/OS Version 2 Release 3 (V2R3)**

---

The following changes are made for z/OS Version 2 Release 3 (V2R3).

#### **New**

- For APAR OA53372, new keyword, BUFLOC, is added to “[\\$CBIO – Control block I/O routine](#)” on page 67.
- BPX1SOC, BPX1CLO, and BPX1MPC are added. See “[\\$BPXCALL – Call omvs services](#)” on page 63.
- Cell types are added to TYPE=celltype. See “[\\$CPOOL – Build, delete, modify, or query a cell pool](#)” on page 82.

- Keyword PTRSIZE= is added to:
  - [“\\$FIFODEQ – Remove an element from a FIFO queue” on page 168](#)
  - [“\\$FIFOENQ – Add an element to a FIFO queue” on page 169](#)
  - [“\\$FIFOGTQ - Dequeue an entire FIFO queue” on page 170](#)
- Keywords, CKPTGRP and CKPTNET, are added for [“\\$JQESERV – User environment JQE services” on page 230](#)
- Keywords, EMAIL= and STOKEN=, are added for email processing. See [“\\$NOTIFY – Send a notify message to a specific user ID and node” on page 275](#).
- Resources are added to the event/resource list if \$HASPECF is specified. See [“\\$POST – Post a JES2 event complete” on page 291](#).
- Optional keyword, PADO, is added for WIDTH. See [“\\$SCAND – Call the \\$SCAN facility display service routine” on page 336](#) and [“\\$SCANTAB – Create a scan table” on page 341](#).
- Keyword, FORMAT, is added. See [“\\$SCANTAB – Create a scan table” on page 341](#).
- CB types are added to the TYPE=list. See [“\\$VERIFY – Verify a control block” on page 393](#).
- Keyword, WTO, is added. See [“\\$STMTLOG – Log an initialization statement” on page 372](#).
- Resources are added to the event/resource list. See [“\\$WAIT – Wait for a JES2 event” on page 399](#).

## Changed

- With APAR OA49945, MSG= is modified for \$RMSGQUE. See [“\\$RMSGQUE – Queue message to JES2” on page 324](#).
- With APAR OA47056, [“\\$QGET – Obtain a job queue element from the JES2 job queue” on page 302](#) is modified.
- With APAR OA50971, [“\\$XMPOST – POST task in another address space” on page 421](#) is modified with new keyword IRB.



---

# Chapter 1. Macro overview

The following JES2 control service programs provide a comprehensive set of services that aid the JES2 processors in performing their respective tasks in an efficient manner without burdening the programmer with needless detail. These services are requested by the processor with JES2 macro instructions. The macros should not be used in code executing outside the control of the JES2 dispatcher unless stated in the description of the individual macro instruction.

- General storage management: Provide for the acquisition and release of JES2 buffers
- Work area management services: Provide for the acquisition and return of work areas that are chained off the processor control element (PCE)
- Virtual page service macros: Provide for the acquisition and release of virtual pages
- Job queue services: Provide for the alteration of job queues
- Direct-access space services: Provide for the allocation and deallocation of JES2 direct-access storage space
- Unit services: Provide for the acquisition and release of JES2 input/output units
- Input/output services: Provide communication with operating system input/output supervisor
- Console services: Provide all communication with the operator and manipulate associated buffer resources
- Time services: Provide for the setting and interrogation of the interval timer
- Synchronization services: Provide synchronization and communication between JES2 processors, the JES2 dispatcher, and the operating system
- System management facilities services: Provide the processors with an interface to the MVS™ SMF routines
- Installation exit services: Provide the \$EXIT macro that is used to define exit points in JES2
- Debug services: Provide facilities for aid in debugging JES2
- Error services: Provide a uniform way of processing detected errors
- Recovery processing aids: Provide macros to aid in recovery processing
- Coding aid services: Provide the JES2 programmer with coding aids not usually available in the operating system, but useful in coding JES2 routines
- Print/punch output services: Provide macros used to define separator pages and create peripheral data information records
- Job control table extension services: Provide macros used to add, expand, locate, and remove extensions to the job control table (\$JCT). Installations can use these extensions to store and transmit job-related information from node to node.
- Job output services: Provide macros used for job output services
- Initialization services: Provide macros to generate initialization statement (and parameter) tables and checkpoint information tables
- Verify services: Provide facilities to build control block verification tables to verify spool-resident control blocks
- Table services: Provide facilities for scanning JES2 initialization statements, dynamically creating control blocks for DCTs, PCEs, and DTEs.
- Miscellaneous services: Provide miscellaneous services such as modify the current JESNEWS data set and switch addressing mode

Some of the above services are provided by inline code expansion wherever the macro instruction is used. The remaining services are provided by routines that are integral parts of the control service programs.

At execution time, the macro expansion passes information to the control program routine to specify the exact nature of the service to be performed. This information is broken down into parameters and, usually, is passed to the routine through general purpose registers called parameter registers.

## How to read syntax diagrams

---

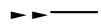
This section describes how to read syntax diagrams. It defines syntax diagram symbols, items that may be contained within the diagrams (keywords, variables, delimiters, operators, fragment references, operands) and provides syntax examples that contain these items.

Syntax diagrams pictorially display the order and parts (options and arguments) that comprise a command statement. They are read from left to right and from top to bottom, following the main path of the horizontal line.

### Symbols

The following symbols may be displayed in syntax diagrams:

Symbol	Definition
--------	------------



Indicates the beginning of the syntax diagram.



Indicates that the syntax diagram is continued to the next line.



Indicates that the syntax is continued from the previous line.



Indicates the end of the syntax diagram.

### Syntax items

Syntax diagrams contain many different items. Syntax items include:

- Keywords - a command name or any other literal information.
- Variables - variables are italicized, appear in lowercase and represent the name of values you can supply.
- Delimiters - delimiters indicate the start or end of keywords, variables, or operators. For example, a left parenthesis is a delimiter.
- Operators - operators include add (+), subtract (-), multiply (\*), divide (/), equal (=), and other mathematical operations that may need to be performed.
- Fragment references - a part of a syntax diagram, separated from the diagram to show greater detail.
- Separators - a separator separates keywords, variables or operators. For example, a comma (,) is a separator.

Keywords, variables, and operators may be displayed as required, optional, or default. Fragments, separators, and delimiters may be displayed as required or optional.

Item type	Definition
-----------	------------

<b>Required</b>	Required items are displayed on the main path of the horizontal line.
<b>Optional</b>	Optional items are displayed below the main path of the horizontal line.
<b>Default</b>	Default items are displayed above the main path of the horizontal line.



# Syntax examples

The following table provides syntax examples.

Table 1. Syntax examples

Item	Syntax example
Required item. Required items appear on the main path of the horizontal line. You must specify these items.	►► KEYWORD — required_item ►►
Required choice. A required choice (two or more items) appears in a vertical stack on the main path of the horizontal line. You must choose one of the items in the stack.	►► KEYWORD — <div>required_choice1 required_choice2</div> ►►
Optional item. Optional items appear below the main path of the horizontal line.	►► KEYWORD — <div>optional_item</div> ►►
Optional choice. A optional choice (two or more items) appear in a vertical stack below the main path of the horizontal line. You may choose one of the items in the stack.	►► KEYWORD — <div>optional_choice1 optional_choice2</div> ►►
Default. Default items appear above the main path of the horizontal line. The remaining items (required or optional) appear on (required) or below (optional) the main path of the horizontal line. The following example displays a default with optional items.	►► KEYWORD — <div>default_choice1 optional_choice2 optional_choice3</div> ►►
Variable. Variables appear in lowercase italics. They represent names or values.	►► KEYWORD — <i>variable</i> ►►
Repeatable item. An arrow returning to the left above the main path of the horizontal line indicates an item that can be repeated.  An arrow returning to the left above a group of repeatable items indicates that one of the items can be selected, or a single item can be repeated.	►► KEYWORD — <div>repeatable_item</div> ►►
Fragment. The   fragment   symbol indicates that a labelled group is described below the main syntax diagram. Syntax is occasionally broken into fragments if the inclusion of the fragment would overly complicate the main syntax diagram.	►► KEYWORD — <div>fragment</div> ►►  fragment ►► — , — required_choice1 — ►►   — , — required_choice2 — <div>, — default_choice   — optional_choice</div> — ►►

## Macro expansion

---

The macro expansion can contain load instructions (LA, L, LH, etc.) that load parameters in parameter registers, and it can contain instructions (LR,...) that load parameter registers from registers loaded by the processor. The processor can also load parameters directly. Registers 1 and 0 are generally used as parameter registers.

Any statement within this manual referencing requirements of caller needing to be in AR ASC mode, or 64-bit mode, also entails that the respective \$AMODE or \$ARMODE macro be used with SYSSTATE=YES or that a SYSSTATE macro has been used.

Each parameter resulting from the expansion of a macro instruction is either an address or a value. An address parameter is a standard 31-bit address. Any exception to this rule will be stated in the individual macro instruction description.

An address parameter is always an effective address. The control service program is never given a 16-bit or 20-bit explicit address of the form D(B) or D(X,B) and then required to form an effective address. When an effective address is to be resolved, it is formed either by the macro expansion or before the macro instruction is issued.

A value parameter is a field of data other than an address. It is of variable length and is usually in the rightmost bits of a parameter register. The value parameter always has a binary format. The leftmost unused bits in the parameter register should contain all zeros. Any exception to this rule is stated in the individual macro instruction description.

Certain value parameters can be placed in a register along with another parameter, which can either be an address or a value parameter. In this case, a value parameter is in other than the rightmost bits. Two or more parameters in the same register are called packed parameters.

Parameters are specified by operands in the macro instruction. An address parameter can result from a relocatable expression or, in certain macro instructions, from an implicit or explicit address. A value parameter can result from an absolute expression or a specific character string. Address and value parameters can both be specified by operands written as an absolute expression enclosed in parentheses. This operand form is called register notation. The value of the expression designates a register into which the specified parameter must be loaded by the processor before the macro instruction is issued. The contents of this register are then placed in a parameter register by the macro expansion.

## Specify JES2 macro instructions

---

The programmer codes an operand on a JES2 macro instruction to specify the exact nature of the service to be performed. Operands are of two types, positional operands and keyword operands.

A positional operand is written as a string of characters. This character string can be an expression, an implied or explicit address, or some special operand form allowed in a particular macro instruction. Positional operands must be written in a specific order. If a positional operand is omitted and another positional operand is written to the right of it, the comma that would normally have preceded the omitted operand must be written. This comma should be written only if followed by a positional operand; it need not be written if followed by a keyword operand or a blank.

In the following examples, EX1 has three positional operands. In EX2, the second of three positional operands is omitted but must still be delimited by commas. In EX3, the first and third operands are omitted; no comma need be written to the right of the second operand.

```
EX1    $EXAMP A,B,C
EX2    $EXAMP A,,C
EX3    $EXAMP ,B
```

A keyword operand is written as a keyword immediately followed by an equal sign and an optional value.

A keyword consists of one through seven letters and digits, the first of which must be a letter. It must be written exactly as shown in the individual macro instruction description.

An optional value is written as a character string in the same way as a positional operand.

Keyword operands can be written in any order, but they must be written to the right of any positional operands in the macro instruction.

In the following examples, EX1 shows two keyword operands. EX2 shows the keyword operands written in a different order to the right of any positional operands. In EX3, the second and third positional operands are omitted; they need not be delimited by commas, because they are not followed by any positional operands.

```
EX1      $EXAMP KW1=X, KW2=Y
EX2      $EXAMP A, B, C, KW2=Y, KW1=X
EX3      $EXAMP A, KW1=X, KW2=Y
```

Certain operands are required in a macro instruction if the macro instruction is to make a meaningful request for a service. Other operands are optional and can be omitted. Whether an operand is required or optional is indicated in the individual macro instruction description.

## Basic notation used to describe macro instructions

JES2 macro instructions are presented in this section using macro instruction descriptions, each of which contains an illustration of the macro instruction format. This illustration is called a format description. An example of a format description is as follows:

```
[symbol] $EXAMP name1-value mnemonic, name2-CODED VALUE, c
              KEYWD1=value mnemonic, c
              KEYWD2=CODED VALUE, c
              KEYWD3=(label, value)
```

Operand representations in format descriptions contain the following elements:

- An operand name, which is a single mnemonic word used to see the operand. For a keyword operand, the keyword is the name. For a positional operand, the name is merely a reference. In the above format description, name1, name2, KEYWD1, and KEYWD2 are operand names.
- A value mnemonic used to indicate how the operand should be written if it is not written as a coded value. For example, addr is a value mnemonic that specifies that an operand or optional value is to be written as either a relocatable expression or register notation.
- A coded value, which is a character string that is to be written exactly as it is shown. For example, RDR is a coded value.
- Parentheses are always required around a list of specifications or values specified for a keyword with multiple values, such as KEYWD3=. These parentheses are optional if only one value is coded or the keyword is allowed to default.

The format description also specifies when single operands and combinations of operands should be written. This information is indicated by notational elements called *metasymbols*. For example, in the preceding format description, the brackets around symbol indicate that a symbol in this field is optional.

## Operand representation

Positional operands are represented in format descriptions in one of two ways:

- By a three-part structure consisting of an operand name, a hyphen, and a value mnemonic, for example: name1-addr
- By a three-part structure consisting of an operand name, a hyphen, and a coded value, for example: name1-RDR.

Keyword operands are represented in format descriptions in one of two ways:

- By a three-part structure consisting of a keyword, an equal sign, and a value mnemonic, for example: KEYWD1=addr
- By a three-part structure consisting of a keyword, an equal sign, and a coded value, for example: KEYWD1=RDR.

The most significant characteristic of an operand representation is whether a value mnemonic or coded value is used; these two cases are discussed next.

## Operands with value mnemonics

When a keyword operand is represented by:

```
KEYWORD=value mnemonic
```

the programmer first writes the keyword and the equal sign and then a value of one of the forms specified by the value mnemonic.

When a positional operand is represented by:

```
name-value mnemonic
```

the programmer writes only a value of one of the forms specified by the value mnemonic. The operand name is merely a means of referring to the operand in the format description; the hyphen simply separates the name from the value mnemonic. Neither is written.

The following general rule applies to the interpretation of operand representations in a format description; anything shown in uppercase letters must be written exactly as shown; anything shown in lowercase letters is to be replaced with a value provided by the programmer. Thus, for a keyword operand, the keyword and equal sign are written as shown, and the value mnemonic is replaced. For a positional operand, the entire representation is replaced.

The value mnemonics listed below specify most of the allowable operand forms that can be written in JES2 macro instructions. Other value mnemonics, which are rarely used, are defined in individual macro instruction descriptions.

- symbol: The operand can be written as a symbol.
- relexp: The operand can be written as a relocatable expression.
- addr: The operand can be written as (1) a relocatable expression or (2) register notation designating a register that contains an address. The designated register must be one of the registers 2 through 12, unless special notation is used.
- addrx: The operand can be written as (1) an indexed or non-indexed implied or explicit address or (2) register notation designating a register that contains an address. An explicit address must be written in the RX form of an assembler language instruction.
- adval: The operand can be written as (1) an indexed or non-indexed implied or explicit address or (2) register notation designating a register that contains a value. An explicit address must be written in the RX form of an assembler language instruction.
- absexp: The operand can be written as an absolute expression.
- rx-addr: The address can be written as one of the following:
  - A register that contains the value
  - An expression that can be used as the second operand of a Load Address (LA) instruction.
- A-type address: The address can be written as any address that is valid in an A-type address constant.
- value: The operand can be written as (1) an absolute expression or (2) register notation designating a register that contains a value.
- text: The operand can be written as a character constant as in a DC data definition instruction. The format description shows explicitly if the character constant is to be enclosed in apostrophes.
- code: The operand can be written as one of a large set of coded values; these values are defined in the macro instruction description.

## Coded value operands

Operands that are not represented in format descriptions by value mnemonics are represented by one or more uppercase character strings that show exactly how the operand should be written. These character strings are called *coded values*, and the operands for which they are written are called *coded value operands*.

A coded value operand results in either a specific value parameter or a specific sequence of executable instructions.

If a positional operand can be written as any one of two or more coded values, all possible coded values are listed in a format description and are separated by vertical strokes indicating that only one of the values is to be used.

## Metasymbols

Metasymbols are symbols that convey information about how to code. Metasymbols denote operands are optional or required when coding macro instructions; they are never written in the coded macro. They show the programmer how and when an operand should be written. The metasymbols used in this section are:

### Metasymbol Meaning and Use

**{ }**

Braces — denote grouping of alternative operands, one of which must be selected. For example:

```
{YES|NO}
```

**[ ]**

Brackets — denote optional operands. Anything enclosed within brackets can be either omitted or written once in the macro instruction. For example:

```
[USE=code]
```

In the example above, the keyword and its operand can either be written or omitted; its use is optional.

**\_**

Underscore — denotes the JES2 default if the particular keyword is not coded. For example:

```
WAIT={YES|NO}
```

In the example above, YES is the default. To override the default, you must code WAIT=NO. The WAIT= keyword is therefore optional; any keyword with a default is enclosed within brackets in the syntax diagrams throughout this chapter, similar to the next example.

Metasymbols are nested in almost any combination throughout the macro instruction descriptions that follow. Whether any set of keywords and operands are optional or required is determined by the outermost set of metasymbols. For example:

```
[,WAIT={YES|NO}]
```

The entire keyword/operand statement is optional, but if you do code the WAIT= keyword, the only valid options are either WAIT=YES or WAIT=NO.

Uppercase operands must be coded as written in the syntax diagrams. Also, punctuation such as commas, parentheses, and single quotation marks are not metasymbols; if present in the syntax diagrams they **must** be coded. Operands in lowercase are **not** to be coded as written; they denote variables that are explained in the description of the particular keyword for the macro instruction.

# Special register notation

Many JES2 macro instruction keywords allow you to code a register as a valid specification. If you do code a register (for example, R0 or R15), be certain to enclose the symbol representing that register in parenthesis. A symbol enclosed in parenthesis is called register notation (for example, (R0) or (R1)).

If an operand of a JES2 macro instruction is written using register notation, the resulting macro expansion loads the parameter contained in the designated register into either parameter register 1 or parameter register 0.

For example, if an operand is written as (R15) and if the corresponding parameter is to be passed to the control program in register 1, the macro expansion would contain the instruction:

```
LR R1,R15
```

Before macro expansion, the processor can load parameter registers; this is called **pre-loading**. When preloading a parameter register, use the JES2 equated symbols for register 0 or 1 (that is, R1 or R0) to indicate to the macro which registers contain values to be passed. If you do not use the JES2 equated symbols, you will cause the generation of an extra instruction.

For example, RONE, an absolute symbol equated to R1, should not be specified on the macro statement if the register required is R1. The macro will not recognize RONE as register 1 and will attempt to load the parameter register R1 from the RONE specification with the following redundant instruction:

```
LR R1,RONE
```

The format description shows whether special register notation can be used, and for which operands. This is demonstrated by the following example:

```
[symbol] $EXAMP {abc-addrx} , {def-addrx}  
            { (R1) }      { (R0) }
```

Both operands can be written in the addrx form, and therefore can be written using register notation. Ordinary register notation indicates that the parameter register should be loaded from the designated register by the macro expansion. The format description also shows that the abc operand can be written as (R1), and the def operand can be written as (R0). If either of these special notations is used, the processor must have loaded the designated parameter register before the execution of the macro instruction.

## Register stability

Usually the following registers cannot be considered stable across a JES2 macro expansion:

- R14
- R15
- R0
- R1

Registers 2-13 are not affected by JES2 macros, unless it is specifically stated in the individual macro instruction description.

## Macro selection table

Table 2 on page 9 summarizes the available JES2 programmer macros by the function they perform. Following [Table 2 on page 9](#) are the individual macro's descriptions, presented in alphabetical order.

<i>Table 2. JES2 Macro Selection Table</i>	
<b>JES2 Service</b>	<b>JES2 Programmer Macros</b>
Checkpoint Services	\$BERTTAB \$CHECK \$CKPT \$DILBERT \$DOGBERT \$DOGWSCQ \$PAIR
Coding Aid Services	\$CALL \$CFSEL \$ENTRY \$MODEND \$MODULE \$PATCHSP \$QUEUE \$RESTORE \$RETURN \$SAVE \$STORE \$SUBIT \$XECBSRV \$RETSAVE \$VFL
Console Services	\$CWTO \$DOM \$FRECMB \$GETCMB \$LOGMSG \$MID \$MSG \$RDIRTAB \$WTO \$\$WTO \$\$WTOR
Debug Services	\$SDUMP \$TRACE
Direct-Access Space Services	\$BLDTGB \$PURGE \$TRACK
Dynamic Service Access Services	\$DCBDYN \$DCTDYN \$DESTDYN \$DTEDYN \$PCEDYN
Error Services	\$DISTERR \$ERROR \$IOERROR \$SYMREC \$SYMTAB
Functional Subsystem Interface Services	\$FSILINK
General Storage Management	\$BFRBLD \$BLDQC \$CPOOL \$DSPSERV \$FRECEL \$FREEBUF \$FREMAIN \$FREQC \$GETBLK \$GETBUF \$GETCEL \$GETHP \$GETMAIN \$GETQC \$QCTGEN \$RETBK
Initialization Services	\$STMTLOG
Input Output Services	\$CBIO \$EXCP \$EXTP
Installation Exit Services	\$EXIT \$ENVIRON \$MODCHK \$MODELET \$MODLOAD
Job Control Table Extension Services	\$JCTXADD \$JCTXEXP \$JCTXGET \$JCTXREM
Job Output Services	##ADD ##ALCHK ##BLD ##BUSY ##CAN ##CHK ##DISPRO ##GET ##GETHDJ ##JWEL ##JOE ##MOD ##POST ##PUT ##REM ##REP ##TJEV
Job Queue Services	\$DOGJQE \$JCAN \$QADD \$QBUSY \$QGET \$QJIX \$QJQE \$QLOC \$QLOCNXT \$QMOD \$QPUT \$QREM
Miscellaneous	\$ALET \$ALESERV \$AMODE \$ARMODE \$BLDMSG \$DEST \$DOGCAT \$DVIDBLD \$GETADDR \$GETASCB \$GETRTN \$JBIDBLD \$JQEJNUM \$MVCL \$PCETERM \$SETIDAW
Networking Services	\$NATGET \$NHDADD \$NHDEXP \$NHDGET \$NHDREM \$NHDXMT
Peripheral Data Definition Block Services and Input/Output Table Services	\$IOTBLD \$PDBBLD \$PDBFIND
Print/Punch Output Services	\$PBLOCK \$PRPUT \$SEPPDIR
Recovery Processing Services	\$ESTAE \$SETRP
Scan Services	\$SCAN \$SCANB \$SCANCOM \$SCAND \$SCANDIA \$SCANTAB
Security Services	\$SEAS

Table 2. JES2 Macro Selection Table (continued)	
JES2 Service	JES2 Programmer Macros
Subsystem Interface Services	\$SSI BEGIN \$SSI END
Subsystem Job Block Services	\$SJB FIND \$SJB LOCK \$SJB REQ
Synchronization Services	\$ACTIVE \$DORMANT \$FRELOCK \$GETLOCK \$POSTQ \$POST \$POST \$QSUSE \$WAIT \$XMPOST
System Management Facility Services	\$GETSMFB \$QUESMFB
Table Pair Services	\$DCTTAB \$DTETAB \$GETABLE \$PCETAB \$TIDTAB \$WSTAB
Time Services	\$STCK \$STIMER \$TTIMER
Unit Services	\$ALLOC \$FREUCBS \$FREUNIT \$GETUCBS \$GETUNIT
Verify Services	\$VERIFY \$VERTAB
Virtual Page Services	\$PGSRVC
Work Area Management Services	\$GETWORK \$REWORK
Work Selection	\$WSSETUP

## Using the \$JCTX macro extension service

The \$JCTX macro extension service allows installations to create extensions to the job control table (\$JCT) control block. These extensions can be SPOOLED for long term storing and transmitting job-related information or local extensions that are for passing information between exits during the current phase of the job. The \$JCTX macro extension service is composed of the following macros:

### Macro Use

#### \$JCTXADD

Creates an extension to the \$JCT based on an installation-specified length and unique identifier. At the time of the add, the requester can specify if the added section is to be SPOOLED or is a local section.

#### \$JCTXEXP

Expands an extension to the \$JCT based on an installation-specified length and unique identifier.

#### \$JCTXGET

Locates an extension to the \$JCT based on an installation-specified unique identifier. By examining the reason code from \$JCTXGET, the requester can determine if the section is a SPOOLED section or a local section.

#### \$JCTXREM

Removes an extension to the \$JCT based on an installation-specified unique identifier.

## Determining the amount of SPOOL space used by SPOOLED \$JCT extensions

The length of a \$JCT extension is limited by the size of the \$JCT control block and an installation's spool buffer size. IBM provides 512 bytes in the spool buffer, in addition to anything beyond the smallest buffer size available (1944 bytes). The buffer size is specified on the BUFDEF= parameter of the SPOOLDEF initialization statement. Use the algorithm in the following figure to determine how much space is used by \$JCT control block extensions within a \$JCT control block:



```
$BUFSIZE (JCT buffer) - JCTFEND-JCTSTART (Fixed $JCT) - JCTRXLEN (space left)
= Space used by JCT extensions in this $JCT
```

Figure 1. Determine the Amount of Spool Space Used by \$JCTX Extensions

The following assembler instructions use this formula:

LH	R1,\$BUFSIZE	Get spool buffer size
SH	R1,=Y(JCTFEND-JCTSTART)	Subtract size of fixed portion of the JCT
SH	R1,JCTRXLEN	Subtract amount of unused space

To determine the amount of space that remains in this \$JCT control block for extensions, specify:

```
LH    R1,JCTRXLEN
```

## Using local \$JCT extensions

Local JCT extensions can be used to store up to 8184 bytes of data associated with the JCT. JES2 does not write this information to SPOOL and this information is deleted when the storage associated with the JCT is freed. Local extensions are intended to pass information between exits (in particular from exits in the USER environment and exits in the JES2 Main Task). Not all exits support using local \$JCT extensions. Support for local \$JCT extensions is documented in each exit.

## Examples of the \$JCTX macro extension service

The following examples illustrate how to use this service to provide job-related information. These examples emphasize how installations can carry the information with the job from node to node across a network and reference the information when needed. Note that these are two simple examples of how the \$JCTX macro extension service can be used at your installation.

### Example 1: Transmitting separator notes through \$JCT extensions

This topic provides an overview of the example provided in the sample exit HASXJECL, which is shipped in SYS1.VnRnMn.SHASSAMP. \$JCTX extensions to the \$JCT control block are created to store one-line notes that appear on the separator page between each job printed. These one-line separator page notes enter the local node through an installation-defined JES2 control statement (/SEPNOTE) with the following syntax:

```
/*SEPNOTE text
```

where 'text' is what the submitter specifies in the job stream to appear on the separator page. Note that this is not an IBM-defined JES2 control statement; it is an example.

If the printer receiving the /\*SEPNOTE statement from the network job header does not print a separator page (SEP=NO on the PRT(nnnn) initialization statement), the notes do not print.

To provide this capability across the network, all nodes must provide the same four exits. However, if you code exits 1 and 4 at the local node only, any jobs entering the system or printed at this node contain the one-line separator note.

The following provides an overview of this function:

1. Before the individual exit points, define the \$JCTX extension values and the NJE header values for the /\*SEPNOTE statements.
2. Exit 4 allows the reader to receive each extension (the /\*SEPNOTE JES2 control statement) and add them to the \$JCT control block in Exit 4
3. Exit 46 allows the job or SYSOUT transmitter to move each extension from the \$JCT control block into a network job header for transmission to another node in Exit 46
4. Exit 47 allows the job or SYSOUT receiver to move each extension from the network job header into the \$JCT at the receiving node in Exit 47

- Exit 1 allows a printer with appropriate work selection values to print the one-line separator page notes.

For a complete understanding of the exit, see the comments in sample exit HASXJECL.

## Example 2: Using \$JCTX extensions to maintain spool compatibility

You can add to an extension while maintaining spool compatibility so that the new and old versions of the \$JCT control block co-exist in the same multi-access spool (MAS) configuration.

By modifying the \$JCTX extensions you are, in effect, modifying the \$JCT control block without having to cold start JES2. If you add fields to the \$JCT data area after the JCTFEND label rather than before it, these modified fields will not be overwritten.

IBM suggests that you convert all \$JCT control block modifications to use \$JCTX extensions. If you add fields to the \$JCT data area directly rather than use the \$JCTX macro extension service, your extensions can be overwritten by other products using the \$JCTX macro extension service.

In the user control block DSECT (\$USERCBS macro):

- Begin by defining a new field to the installation-defined "accounting" extension at the end of the extension so the offsets of other fields do not change.

```

:
JCTX      DSECT      Set to JCTX DSECT
          ORG        Set location counter to the correct origin
JCTXACCT  DS         CL20      Installation account information
JCTXACT2 DS         CL8       Additional accounting information
JCTXLEN1  EQU        *-JCTX    Length of the extension
:
```

- Then, define a new field to the installation-defined section of the network job header at the end of the extension field so the offsets of these fields do not change.

```

:
NJHU      DSECT      Installation section
          ORG        NJHUCODE+L NJHUCODE
NJHU$MD1  EQU        B'11101101'
NJHUACCT  DC         CL20      Installation account information
NJHUACT2 DC         CL8       Additional accounting information
NJHULEN1  EQU        *-NJHU    Length of the accounting section
:
```

## Sample Exit 6 for spool compatibility

Write an Exit 6 that creates the second extension and ensures that it is long enough to contain the new 8-byte field of additional accounting information. This 8-byte field was added during job conversion.

**Note:** If the section already exists and is long enough, JES2 puts the address there.

- Use the \$JCTXGET macro to locate the extension to the \$JCT control block that had been added earlier:

```

:
          $JCTXGET JCT=JCT,      Macro to locate the
          TYPE='ACCT',          $JCTX extension to the
          MOD=1,                $JCT control block.
          NOTFOUND=X6ADDIT,     Add the extension if
          ERRET=X6SKIP          it is not found
                                Return to caller
                                for all other errors
:
```

- Determine the address of the extension and identify the register (in this case, base register 5) as the beginning of the extension address.
- If JES2 finds the extension, use the \$JCTXEXP macro to expand the extension so it can contain the new 8-byte accounting field. Then move the accounting field from the \$JCTX into the network job header, as shown in the previous step:

```

:
$JCTXEXP JCT=JCT,          Expand the extension
        TYPE='ACCT',
        MOD=1,
        LENGTH=JCXXLEN1,
        NOSPACE=X6SKIP,    Return to the caller
                                if there is no room for
                                the expansion
        ERRET=X6SKIP,      Return to the caller
                                for all other errors
        OKRET=X6MOVE       Move the extension into
                                the network header
:

```

4. If JES2 did not find the extension, use the \$JCTXADD macro to add the entire extension. Note that the 'JCXXLEN1' LENGTH= value allows you to include the 8-byte accounting field.

```

:
X6ADDIT    $JCTXADD JCT=JCT,      Add 8-byte field onto
        TYPE='ACCT',          the existing extension
        MOD=1,
        LENGTH=JCXXLEN1,
        NOSPACE=X6SKIP,      Return to caller
                                if there is no room for
                                the expansion
        ERRET=X6SKIP         Return to caller
                                for all other errors
:

```

5. Once you have returned from Exit 6, provide the label (**X6MOVE**) where successful return codes should branch so the expansion is moved into the network header, and the label (**X6SKIP**) where unsuccessful return codes should branch so they can return to the caller.

```

:
X6MOVE      MVC    ACTINFO,JCXXACT2  Move the expansion
:                                          into the network header
:
X6SKIP      DS     OH                Return to caller
:

```

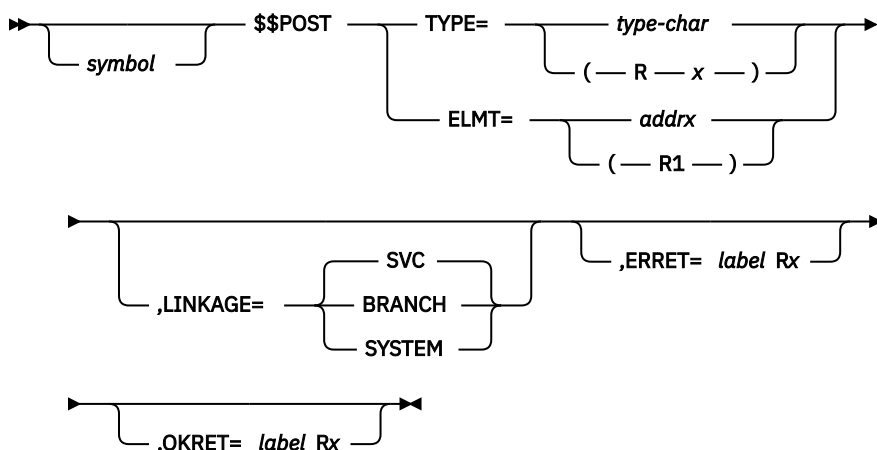


## Chapter 2. JES2 programmer macros

### \$\$POST – Post a JES2 event complete from another task

Use \$\$POST to post specific JES2 processors or resources by setting indicators that cause JES2 processors to begin executing. \$\$POST is for use by routines running in any JES2 environment other than the main task environment. \$\$POST is also for use by asynchronous MVS exit routines associated with the JES2 main task (for example, timer exits and I/O exits).

#### Format description



#### TYPE=

Specifies the resource that is to be posted. You can specify the resource characters (which appear below) or you can specify register notation (R2-R12). If you specify register notation, the register must contain a resource number (0 through 63).

If you specify resource characters, you must specify one of the following:

#### ABIT

Waiting for the next dispatcher cycle

#### ALICE

PCE waiting for warm start to complete

#### ALOC

A dynamic allocation has completed

#### ARMS

Automatic restart manager support services

#### BUF

A JES2 buffer has been released

#### CCAN

Cancel JOB/TSU/STC in conversion

#### CKPT

A JES2 checkpoint write has completed

#### CKPTL

Looking for a CKPT read

#### CKPTP

A checkpoint cycle has completed

**CKPTW**

A JES2 checkpoint should be written

**CMB**

A console message buffer has been released

**CNVT**

A converter has been released

**DAWN**

Post PCEs waiting for work notifications

**DILBERT**

PCEs waiting for \$DILBERT requests

**EOM**

Post PCEs waiting for End Of Memory events

**FSS**

A functional subsystem has completed FSS-level processing

**GENL**

Provides a method of communication from one processor control element (PCE) to another. It does not provide serialization between the PCEs. You must ensure the condition of the waiting PCE is satisfied before it is posted. Frequent use of the GENL resource name will have a severe impact on your installation's performance.

**HOMOG**

PCEs waiting for JESplex version change

**HOPE**

An output processor has been released

**IMAGE**

A UCS or FCB image has been loaded

**IRCLEAN**

Internal reader cleanup needed

**JCMD**

A JES2 job queue element has been marked for cancel (\$C) or restart (\$E) processing

**JOB**

A JES2 job queue element has changed status

**JOE**

A JOE has been released

**JOT**

A JES2 job output element has changed status

**LOCK**

A lock has been released

**MAIN**

Storage is available

**MFMT**

PCEs waiting for SPOOL mini-format conversion

**NRM**

PCEs checking the availability of NJE devices and initiating their starts appropriately.

**PCETM**

Waiting for resource manager to detach PCE

**PSO**

A process SYSOUT request has been queued for the JES2 PSO processors

**PURGE**

A JES2 job queue element (JQE) has been placed on the purge queue

**PURGS**

Purge resources from \$PURGER have been released

**RMWT**

Waiting for resource manager to finish processing

**RSV**

A JES2 RESERVE has been satisfied

**SPI**

PCEs waiting for SYSOUT API (SAPI) requests

**SPIN**

A spin data set has been created

**SMF**

AN SMF buffer has been released

**TRACK**

A track group from the JES2 spooling data set has been released

**UNIT**

A device control table has been released

**WARM**

Warm processor is waiting for work

**XMITJOB**

A JES2 job queue element (JQE) has been placed on the \$XMIT queue to be transmitted to another node.

**value**

An installation-defined dispatcher resource name or number

**ELMT=**

Specifies the address of the element where the event indicator is to be set. Symbolic names for these indicator elements are as follows:

- CCTASYNC – Post asynchronous I/O processor
- CCTCKPTP – Post checkpoint processor
- CCTCOMM – Post command processor
- CCTENFP – Post ENF listen processor
- CCTJOB – Post execution processor
- CCTJQRP – Post JQE request processor
- CCTMLLM – Post line manager
- CCTOFFM – Post SPOOL offload processor
- CCTRCP – Post remote console processor
- CCTSPOOL – Post spool manager
- CCTSSPCE – Post SJF services processor
- CCTTIMER – Post timer processor
- CCTTRPCE – Post trace logger
- CCTXSTIM – Post time excession processor

The corresponding processor control elements are posted by the JES2 dispatcher on recognizing the post elements line in \$\$POST.

If you use register notation, the designated register must be loaded with the address of the element before executing this macro. Do not use register 2 for this address.

**ERRET=**

Specifies a label to be branched to or a register to be branched on if JES2 returns a non-zero return code in R15. This parameter is optional.

**OKRET=**

Specifies a label to be branched to or a register to be branched on if JES2 returns a zero return code in R15. This parameter is optional.

**LINKAGE=**SVC**|**BRANCH**|**SYSTEM

Specifies the type of linkage JES2 is to use when it issues the MVS POST macro instruction. The requirements for specifying each type of linkage depend on from which address space the \$\$POST is issued.

When the \$\$POST is issued from the JES2 address space:

**Linkage Type  
Requirements****BRANCH**

The \$\$POST caller must be in primary ASC mode and must hold the local lock.

**SVC**

None

**SYSTEM**

The \$\$POST caller must not hold any locks.

When the \$\$POST is issued from a non-JES2 address space:

**Linkage Type  
Requirements****BRANCH**

If the \$\$POST caller holds the local lock, the caller must be in the home address space. If the \$\$POST caller does not hold the local lock, the caller can be in any address space.

**SVC**

The \$\$POST caller must be in task mode and in primary ASC mode.

**SYSTEM**

The \$\$POST caller must be enabled, unlocked, and in primary ASC mode.

**Note:**

1. The execution of this macro requires registers 0, 1, 2, 11, and 15.
2. This macro instruction should not be used when executing code that runs under control of the main JES2 task program request block.
3. Either TYPE or ELMT operands must be specified.

**Environment**

- Subtask, user, and functional subsystem (HASPFSM).
- WAIT cannot occur.

**\$\$WTO – JES2 subtask write to operator**

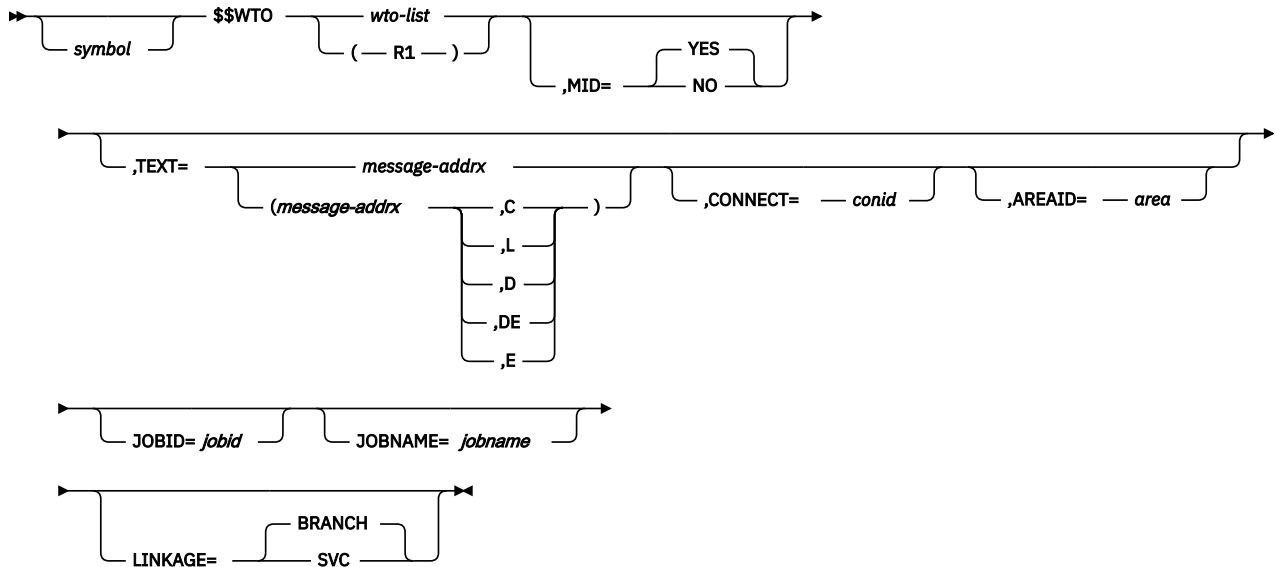
---

Use \$\$WTO to initiate the display of an operator message from a JES2 subtask or during JES2 initialization or termination. The message is issued through an MVS execute-form WTO macro after supplying the JES2 command ID character.

\$\$WTO stores the message text within the message area; therefore, your program becomes non-reentrant after using this macro. Your program remains reentrant if the message area is acquired (using \$GETMAIN) and refreshed each time the macro is issued.



## Format description



### wto-list

Specifies the address of a list-form MVS WTO message. If TEXT= is also specified, it is assumed that the list form WTO also specified TEXT=. If TEXT= was not specified, it is assumed that the list form WTO contains the text of the message to be issued. If descriptor code 1, 2, 3, or 11 was specified, the DOM ID is returned in register 1. If a multi-line WTO was specified, the connect ID is returned in register 1.

### AREAID=

Specifies the console area ID. Only valid if TEXT= was specified

### CONNECT=

If this is not the first line of a multi-line WTO, specifies the MLWTO connect ID. Only valid if TEXT= was specified.

### JOBID=

Specifies the job id to be associated with the message. This parameter is optional.

### JOBNAME=

Specifies the job name to be associated with the message. This parameter is optional.

### LINKAGE=

Specifies whether BRANCH or SVC linkage is to be used to enter the WTO service. The default is LINKAGE=SVC.

### MID=

Specifies whether the message has a message ID and therefore whether or not to add the JES2 COMCHAR to the start of the message. Default is YES.

### TEXT=

Option keyword which specifies the address of the text of the operator message. The address can be in a register (2-12) or be the name of a field. Use of this keyword implies that the WTO MF=L specified TEXT= and that it generates an extended WPL.

The address specified by TEXT= will be placed in the WPL pointed to through the wto-list parameter.

TEXT is the address of a half-word length followed by the message text. The length does not include the length of the half-word.

A second optional value is the line type for multi-line WTOs. Valid values are C, L, D, DE, and E. See [z/OS MVS Programming: Assembler Services Reference IAR-XCT](#) for complete descriptions.

## Environment

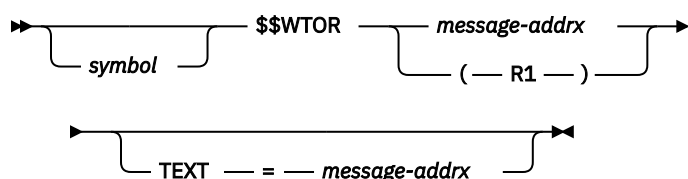
- Subtask or main task.
- \$WAIT cannot occur.

## \$\$WTOR – JES2 subtask write to operator with reply

Use \$WTOR to initiate the display of an operator message, requiring a reply, from a JES2 subtask. The message is issued using an MVS execute-form WTOR macro instruction after supplying the JES2 command ID character.

\$\$WTOR stores the message text within the message area; therefore, your program becomes non-reentrant after using this macro. Your program remains reentrant if the message area is acquired (using \$GETMAIN) and refreshed each time the macro is issued.

## Format description



**message**

Specifies the address of a list-form MVS WTOR message. If register notation is used, the address must be loaded into the designated register before execution of this macro instruction. The DOM ID is returned in register 1.

**Note:**

1. From JES2 subtasks, HASPINIT and HASPTerm, it is the responsibility of the issuer of this macro instruction to issue a WAIT macro instruction, the ECB of which will be posted when the operator has replied to the message.
2. From the main task it is the responsibility of the issuer of this macro instruction to issue a \$WAIT with the XECB option.

**TEXT=**

Specifies the address of the text of the operator message. The address can be in a register (2-12) or be the name of a field. Use of this keyword implies that the WTOR MF=L generates an extended format parameter list.

As in the MVS WTOR macro, TEXT= is the address of a half-word length followed by the text of the WTOR. The length does not include the length of the half-word.

TEXT= is an optional keyword. If not provided, the WTOR MF=L parameter list is assumed to already include the text for the message.

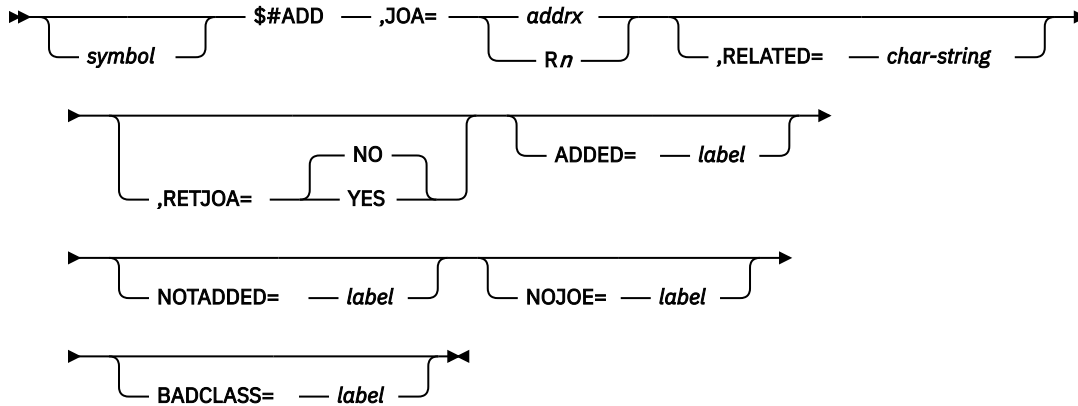
## Environment

- Subtask.
- Main task (during JES2 initialization and termination).
- \$WAIT cannot occur.

### **##ADD – Add a work/characteristics JOE pair to the JOT**

Use **\$#ADD** to add a job output element (JOE) to the appropriate job output table (JOT) queue and add the characteristics JOE to the characteristics queue.

## Format description



### JOA=

Specifies the address or a register (R2-R10) that contains the address of the artificial JOE (JOA) whose work JOE will be added to the JOT. Also the characteristics JOE that is contained in the JOA will be merged into the characteristics queue.

#### Note:

1. The queue to which the work JOE is added is determined by the current class of the JOECURCL and JOEROUT fields of the JOE or by the offload status in the JOEFLAG2 field.
2. If the JOECURCL of the work JOE is not valid, ##ADD issues a \$DISTERR message, DISASTROUS ERROR AT SYMBOL CHKCLDST IN CSECT HASPJOS, unless it is apparent that the JOT has been corrupted. In this event, ##ADD terminates JES2 with a \$ERROR, CATASTROPHIC ABEND J07 INVALID SYSOUT CLASS FOUND.
3. When ##ADD returns control to the caller, register 1 points to the JOE when RETJOA=NO is specified. Otherwise, when RETJOA=YES is specified, register 1 points to the JOA. ##ADD does not alter the setting of the JOE's busy bits. The call should set these bits to the appropriate value.

### RELATED=

Specifies a character string used to self-document this macro call. Any specification type value for macro keywords can be used here. This field is useful for documenting the inline pairing of ##ADD and ##REM macro calls.

### RETJOA=

Specifies whether or not the address of an update mode JOA or the address of the work JOE is returned in register 1 when ##ADD was successful.

#### YES

Indicates that the address of an update mode JOA should be returned.

#### NO

Indicates that the address of the work JOE should be returned. This is the default.

### ADDED=

Specifies optional label to go to if the add was successful.

### NOTADDED=

Specifies optional label to go to if the add failed (RC>0). If NOTADDED= is specified, NOJOE= and BADCLASS= are not allowed.

### NOJOE=

Specifies optional label to go to if the add failed because there are no JOEs (RC=4). If NOJOE= is specified, NOTADDED= is not allowed.

### BADCLASS=

Specifies optional label to go to if the add failed because the SYSOUT in the prototype JOE was not valid (RC=8). If NOJOE= is specified, BADCLASS= is not allowed.

## Return codes

The following return codes (decimal) are returned in register 15.

### Return Code Meaning

**0**

The service was successfully performed.

**4**

The JOT is full; the request must be tried again later.

**8**

An invalid SYSOUT class is encountered; the JOE will not be added.

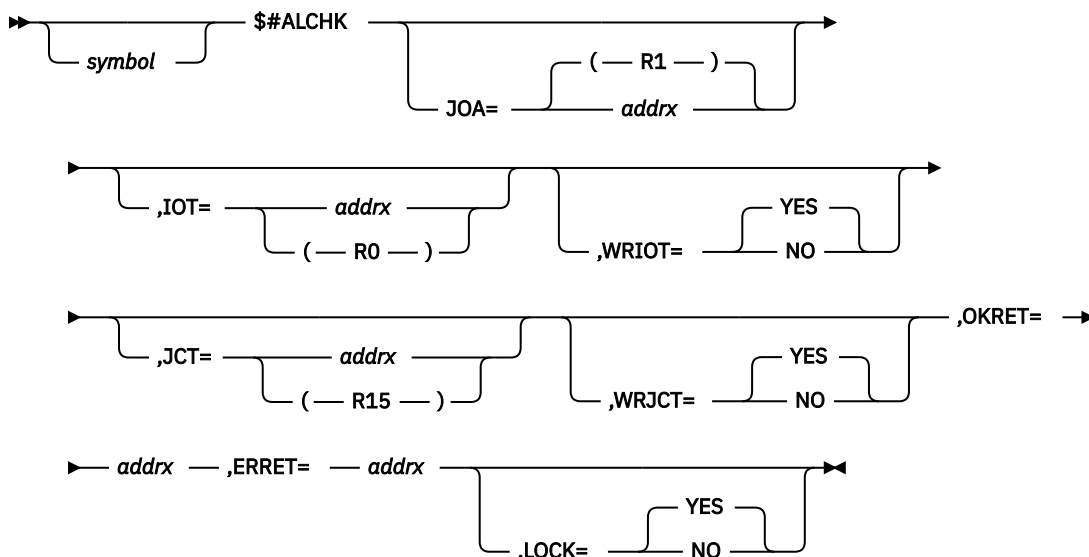
## Environment

- Main task.
- \$WAIT can occur.

## ##ALCHK – Obtain a spool record for output checkpointing

Use ##ALCHK to obtain a spool record for output checkpointing.

## Format description



### JOA=

Specifies the address of an update mode JOA. The JOA represents the work JOE that requires a spool record for output checkpointing. If register notation is used, the designated register must contain the address of the JOA before the execution of the macro. If JOA= is omitted, JES2 assumes that register 1 contains the address.

**Note:** The JOA must be an update mode JOA.

### IOT=

Specifies the address of the IOT that is to be used for allocating the spool. If register notation is used, the designated register must contain the address of the IOT before the execution of the macro. If this operand is omitted, the IOT is read from the spool. An indication is set in the generated inline parameter list whether the IOT was passed.

- Main task.
- \$WAIT can occur.

Use \$#BLD to format a pair of work and characteristics job output elements (JOEs) in the provided work area.

$\text{symbol} \rightarrow \text{\$BLD} \rightarrow \text{JOA} = \text{addrx} \text{ (--- Rn ---)} \rightarrow \text{,PDB} = \text{addrx} \text{ (--- R0 ---)}$   
 $\text{,JQE} = \text{addrx} \text{ (--- R15 ---)}$

**JOA=**  
Specifies the address of the work area that will be formatted into an artificial JOE (JOA). The artificial JOE (JOA) contains the work JOE, the characteristics JOE, the JOE extension (JOX), and BERT fields. If register notation is used, the address must be loaded into the designated register before the execution of this macro instruction.

**Pddb=**  
Specifies the address of the peripheral data definition block (Pddb) whose contents are used to format the work and characteristics JOEs. If register notation is used, the address must be loaded into the designated register before the execution of this macro instruction.

**JQE=**  
Specifies the address location of the JQE to which the PDDb belongs. The location is specified as the address of the JQE from the start of the job queue. The \$DOGJQE service may return a real or artificial JQE. An artificial JQE consists of the base JQE, the JQX, and the additional fields defined in the JQA. If an address is used, it specifies the address of a fullword whose two right-most bytes contains the JQE address.

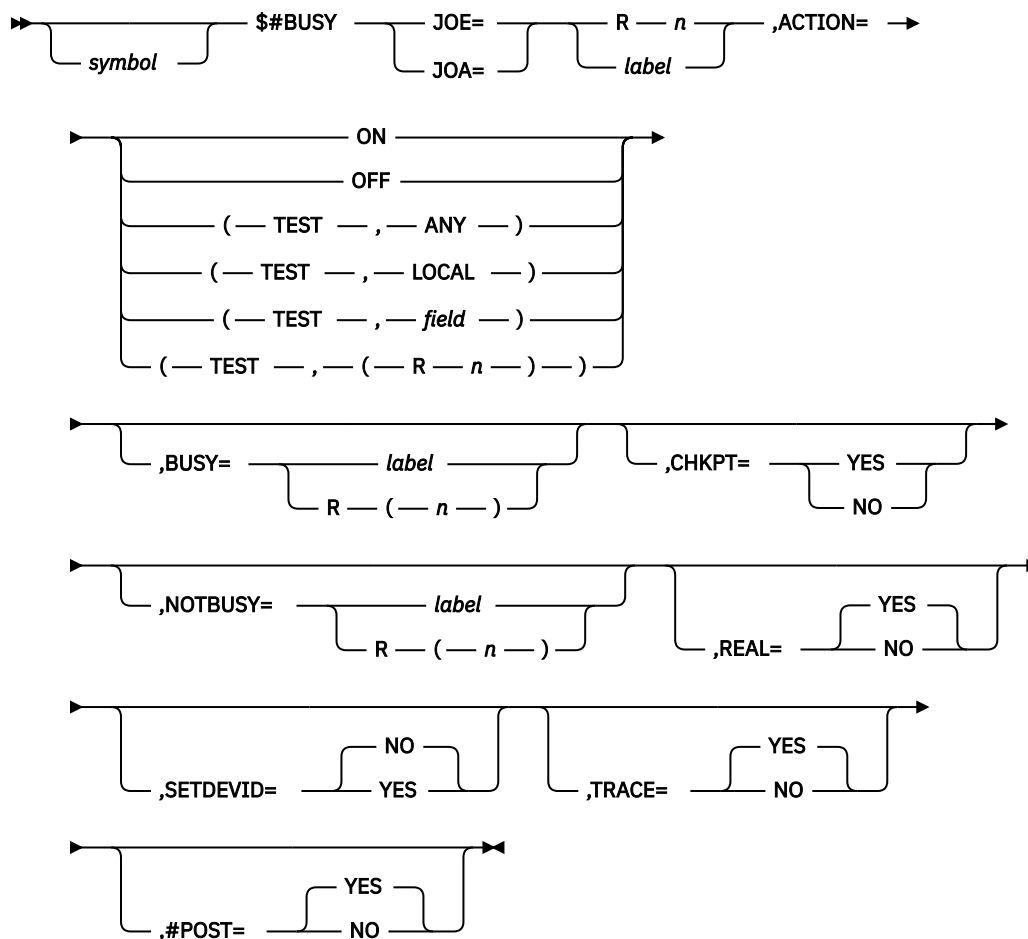
## Environment

- Main task.
- \$WAIT can occur.

## **##BUSY – Set or test the busy system indicator of a JOE**

Use \$#BUSY to set or test the busy system indicator for a job output element (JOE).

## Format description



**JOE=**

Specifies the address of the JOE or JOA whose busy indicator is to be set or tested. If you use register notation, provide the address of the JOE or JOA in the specified register. If you specify a label, that label is the address of the JOE or JOA. JOE= is mutually exclusive with JOA=.

**Note:** If a read mode JOA is passed on JOE=, this service automatically refreshes the JOA before returning to its caller.

**JOA=**

Specifies the address of the JOA whose busy indicator is to be set or tested. If you use register notation, provide the address of the JOA in the specified register. If you specify a label, that label is the address of the JOA. JOA= is mutually exclusive with JOE=.

**Note:** If a read mode JOA is passed on JOA=, this service automatically refreshes the JOA before returning to its caller.

**ACTION=**

Specifies whether the busy indicator for this JOE is to be set on (ON), turned off (OFF), or tested (TEST).

**ON**

Indicates that this member is processing this element. If REAL=YES is also specified, an update mode JOA is required. The \$\$BUSY service checkpoints the busy state before returning to its caller.

**OFF**

Indicates that this element is not being processed by any members. If REAL=YES is also specified, a real JOE, a read mode JOA, or an update mode JOA might be passed, unless CHKPT=NO is specified. If CHKPT=NO is specified, an update mode JOA must be passed.

**(TEST,ANY)**

Indicates that the JOE/JOA should be tested to determine if the JOE is busy on any member of the MAS.

**(TEST,LOCAL)**

Indicates that the JOE/JOA should be tested to determine if the JOE is busy on this member of the MAS.

**(TEST,field)**

Indicates that the JOE/JOA should be tested to determine if the JOE is busy on the member of the MAS whose member number is specified in the indicated 1-byte field.

**(TEST,(Rn))**

Indicates that the JOE/JOA should be tested to determine if the JOE is busy on the member of the MAS whose member number is specified in the indicated register.

**BUSY=**

Specifies a label or register to which to branch if the JOE or JOA is busy on the particular member of the MAS.

BUSY= is only valid if you also specify ACTION=(TEST,...).

**CHKPT=**

Specifies whether (YES) or not (NO) this service checkpoints the JOE or JOA changes. This parameter is valid only when ACTION=OFF and REAL=YES are specified. The default is CHKPT=YES.

**NOTBUSY=**

Specifies a label or register to which to branch if the JOE or JOA is **not** busy on the particular member of the MAS.

NOTBUSY= is only valid if you also specify ACTION=(TEST,...).

**REAL=**

Specifies whether this is a real JOE within the JES2 checkpoint data set (YES) or a prototype JOE in a work area (NO).

**YES**

The JOE= parameter specifies either a real work JOE, a read mode JOA, or an update mode JOA. REAL=YES can only be specified in the JES2 main task environment.

**NO**

The JOE= parameter specifies a JOE that is a prototype to be created, or a JOE that is in a checkpoint version. If the specified JOE is a checkpoint version, do not use ACTION=ON or ACTION=OFF.

TRACE=YES and REAL=NO are mutually exclusive.

**SETDEVID=**

Specifies how to set the JOEDEVID field.

**value**

Specifies a 3-byte value that JOEDEVID will be set to.

## **##CAN**

### **YES**

Indicates that JOEDEVID will be set from PCEDCT.

### **NO**

Indicates that JOEDEVID will not be set. This is the default.

### **TRACE=**

Specifies whether (YES) or not (NO) this modification to the busy indicator is to be traced by the SYSjes2 component trace. See [z/OS MVS Diagnosis: Tools and Service Aids](#) for further information concerning SYSjes2 component tracing.

### **YES**

Indicates that tracing is set on for this ##BUSY call.

TRACE=YES and REAL=NO are mutually exclusive.

### **NO**

Indicates that tracing is set off for this ##BUSY call.

TRACE=YES and REAL=NO are mutually exclusive.

### **Note:**

1. TRACE= is only valid if you also specify either ACTION=ON or ACTION=OFF.
2. IBM suggests that you do not turn SYSjes2 tracing off. If JES2 encounters a problem related to ##BUSY services, the data obtained from this trace can significantly aid debugging procedures.

### **##POST=**

Specifies whether or not a ##POST of the JOE will be done.

### **YES**

Indicates a ##POST of the JOE will be done.

### **NO**

Indicates a ##POST of the JOE will not be done. This is the default.

**Note:** ##POST= is only valid if you also specify ACTION=OFF.

## **Environment**

- Main task.
- \$WAIT or WAIT cannot occur.

### **Note:**

1. On return from the ##BUSY routine, register 15 will contain a 0 (zero) if you specified ACTION=ON or ACTION=OFF. JES2 provides no return codes for ACTION=(TEST,...).
2. Register usage:
  - ##BUSY uses registers R0, R1, R14, and R15 as work registers.
  - On entry, ##BUSY requires that R11 contain the address of the HCT.

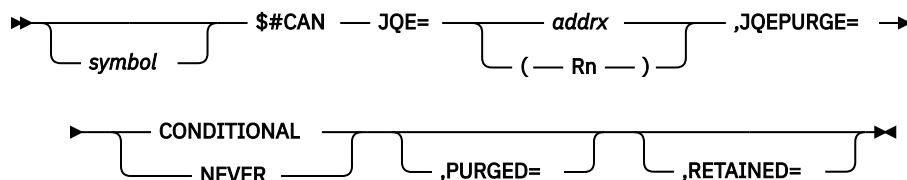
## **##CAN – Cancel all work items not currently being processed for a specific job**

---

Use ##CAN to remove from the JOT all available work items for a job. Work items removed are not processed by any output processor.



## Format description



### JQE=

Specifies the address of the job queue element for which all JOT entries are to be purged.

**Note:** The specified job is purged from the system if all of its output requirements are removed and its current queue position is \$HARDCPY.

### JQEPURGE=

Specifies whether the JQE should be put on the purge queue or not if it is discovered that there are no more JOEs left for the JQE. The default is CONDITIONAL.

#### CONDITIONAL

The JQE is put on the purge queue if it is discovered that there are no more JOEs left for the JQE.

#### NEVER

The JQE is not put on the purge queue if it is discovered that there are no more JOEs left for the JQE.

### PURGED= (Optional)

Specifies a label to be branched to or a register to be branched on if the JQE was purged.

### RETAINED= (Optional)

Specifies a label to be branched to or a register to be branched on if the JQE was not purged.

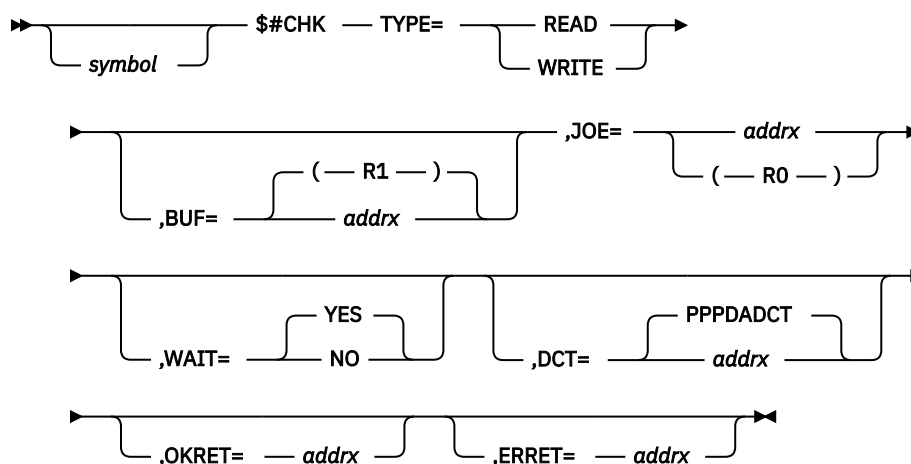
## Environment

- Main task.
- \$WAIT can occur.

## ##CHK – Process print/punch checkpoint spool I/O

Use ##CHK to process print/punch checkpoint spool I/O.

## Format description



**TYPE=**

Specifies whether the operation is a checkpoint read or write. The read or write indication is placed in an inline parameter list (CHK1RD for read and CHK1WR for write). This operand must be specified or an error occurs at assembly time.

**BUF=**

Specifies the address of the checkpoint I/O buffer. If register notation is used, the designated register must contain the address of the buffer. If this operand is omitted, BUF=(R1) is assumed.

**JOE=**

Specifies the address of the work JOE or JOA associated with the spool I/O. If register notation is used, the designated register must contain the address of the work JOE before execution of the macro. This keyword is required.

**Note:** If a read mode JOA is passed on JOE=, this service automatically refreshes the JOA before returning to its caller.

**WAIT=**

Specifies whether to wait for the spool I/O to complete and whether to set a return code. WAIT=YES indicates to wait for the I/O to complete and to set a return code. WAIT=NO indicates to not wait for the I/O to complete and to not set a return code. If this operand is omitted, WAIT=YES is assumed.

**Note:** Specifying WAIT=NO nullifies the use of both the OKRET= and ERRET= keywords.

**DCT=**

Specifies the DCT address needed to perform the spool I/O. If this operand is omitted, PPPDADCT is used.

**OKRET=**

Specifies the address of a routine that is to receive control if the return code is zero.

**Note:** Specifying WAIT=NO nullifies the use of OKRET.

**ERRET=**

Specifies the address of an error routine that is to receive control if the return code is not zero.

**Note:** Specifying WAIT=NO nullifies the use of ERRET. Also, ERRET take precedence over OKRET when both operands are specified.

## Environment

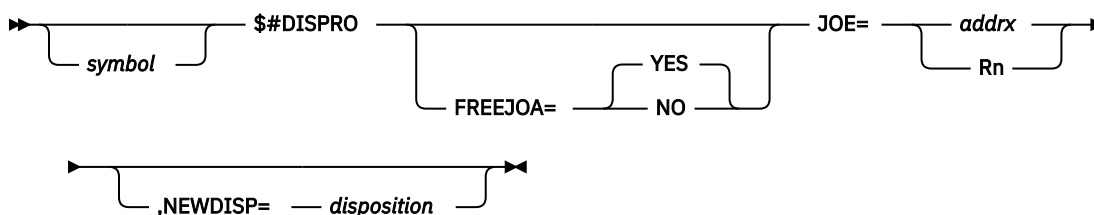
- Main task.
- \$WAIT can occur.

## ##DISPRO – Process JOE disposition

Use ##DISPRO to specify a new disposition for a JOE. If a new disposition is not specified, the JOE's disposition will be processed as follows:

- If the disposition is OUTDISP=KEEP, the JOE's disposition will be altered to OUTDISP=LEAVE.
- If the disposition is OUTDISP=WRITE, the JOE will be removed from the queue.
- If the disposition is OUTDISP=LEAVE or OUTDISP=HOLD, the JOE's disposition will not be modified.

## Format description



**FREEJOA=**

Specifies optional actions for the \$\$DISPRO service if this macro is called with an update mode JOA. YES is the default.

**YES**

Indicates that the following actions should be done:

- Write the JOA changes to checkpoint.
- Free the JOA of the caller and all user stack elements.
- Release the BERT lock.

**NO**

Indicates that any changes made by this service are written to checkpoint.

**JOE=**

Specifies the address or a register that contains the address of the JOE or JOA whose disposition is to be checked or changed. If you specify a register, you cannot use R0.

**NEWDISP=**

Specifies the new disposition for the JOE.

Valid output dispositions are:

**HOLD**

Hold the output. JES2 does not process the output until you either change the disposition to WRITE or KEEP, or release the output. When the output is released, the disposition changes to WRITE.

**KEEP**

Process the output and then keep a copy of it on spool. After processing, the disposition of this output becomes LEAVE.

**LEAVE**

JES2 does not process the output until you change the disposition to WRITE or KEEP, or release the output. When the output is released, the disposition changes to KEEP.

**PURGE**

Purge the output immediately.

**WRITE**

Process the output then purge it.

A register can also be used, if the output group disposition equates found in \$HASPEQU are in the low order byte of the register. You cannot use register 1.

## Environment

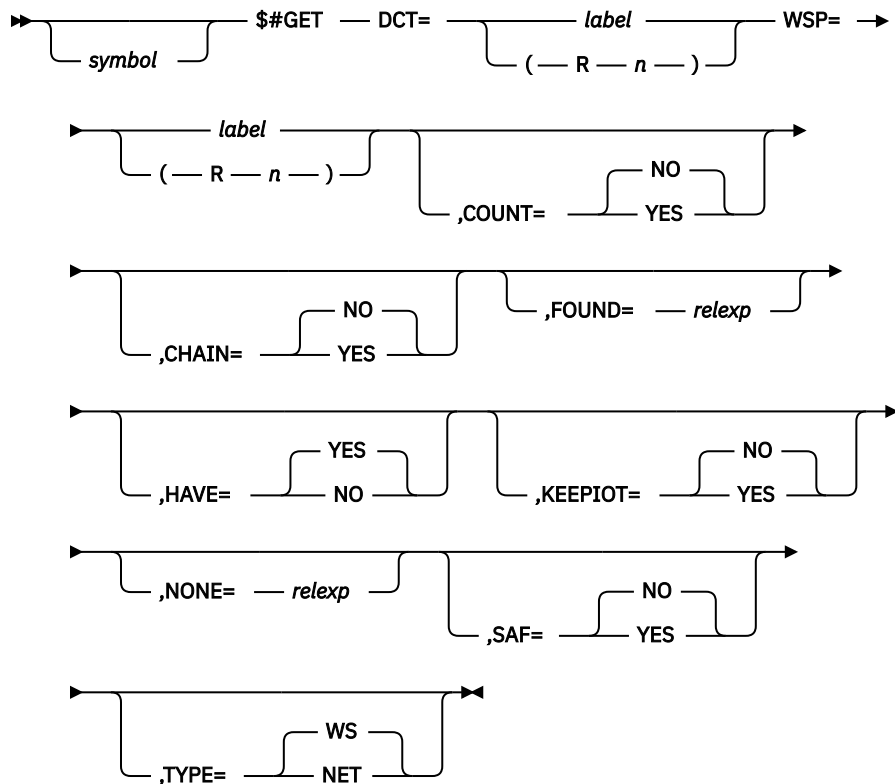
- Main task.
- \$WAIT can occur.

## \$\$GET – Search the JOT class queues for an output element which matches the requesting specification

---

Use \$\$GET to search the JOT for output work.

## Format description



### **DCT=**

Specifies a register (R2-12) or label of the field containing the address of the JES2 device control table (DCT).

DCT= is mutually exclusive with WSP=.

### **CHAIN=**

Specifies either that all (YES) eligible job-related JOEs are to be chained to the transmitter chain and returned to the caller, or only the first (NO) eligible JOE is to be returned to the caller.

CHAIN=YES is mutually exclusive with HAVE=NO.

### **COUNT=**

Specifies whether (YES) or not (NO) JES2 should count the pages or lines or JOEs which match the selection criteria.

COUNT=YES is mutually exclusive with CHAIN=, FOUND=, HAVE=, KEEPLOT=, NONE=, and SAF=.

### **FOUND=**

Specifies a label or address in a register to branch to if a selectable JOE is found.

### **HAVE=**

Specifies that if a selectable JOE is found it is not to be assigned to the requester (NO), or if a selectable JOE is found it is to be assigned to the requester (YES).

HAVE=NO is mutually exclusive with CHAIN=YES.

### **KEEPLOT=YES|NO**

Specifies whether the IOT buffer has been passed to \$#GET by the caller. If KEEPLOT=YES, JES2 will not issue a \$GETBUF or \$FREEBUF for the IOT buffer. Your routine must place the address of the IOT buffer in PCEBUFAD.

KEEPLOT=YES is mutually exclusive with SAF=NO.

**NONE=**

Specifies a label or an address in a register to branch to if there are no selectable JOEs found.

**SAF=**

Specifies whether (YES) or not (NO) JES2 will perform a security authorization facility (SAF) check at this time. SAF=NO causes the SAF check to be deferred until a later time.

SAF=NO is mutually exclusive with KEEPIOT=YES.

**TYPE=**

Specifies that for this ##GET call, either the network queue (NET) is searched or the work selection (WS) algorithm is used.

If you specify TYPE=NET, you must also specify DCT=.

**WSP=**

Specifies a register (R2-12) or label of the field containing the address of the work selection parameter list (WSP) which JES2 loads into R1.

WSP= is mutually exclusive with DCT=.

If you specify TYPE=NET, you must also specify DCT=.

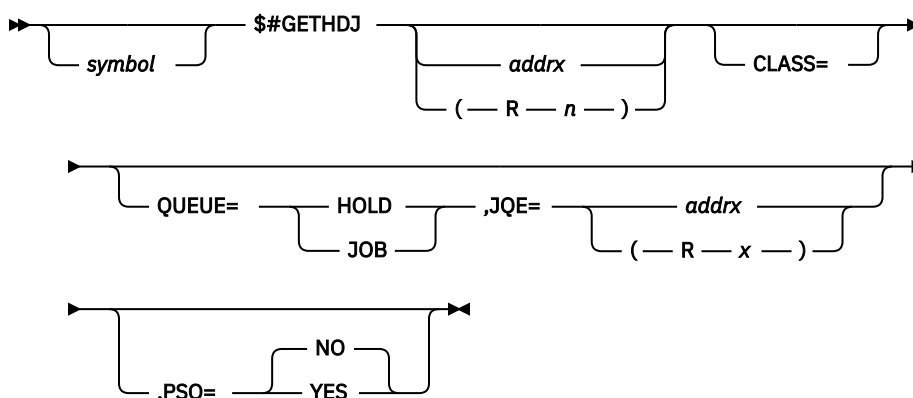
## Environment

- Main task.
- \$WAIT can occur.

## ##GETHDJ – Get held JOE

Use ##GETHDJ to search an output queue to find the held JOE for a job's output.

## Format description



**CLASS=**

Specifies the address of a class list. The class list can have a maximum of eight classes. If less than eight classes are specified, then the last class should be delineated by a blank. If CLASS is specified, ##GETHDJ will do additional checking to ensure the JOE class matches one of the classes in the list. CLASS is an optional keyword. There is no default value. It is only honored when PSO=YES is also coded.

**QUEUE=**

Specifies the queue to be searched.

**HOLD**

Specifies that the HOLD queue should be searched for the first JOE. ##GETHDJ returns in register 1 the address of the JOE. If there is no held JOE, register 1 contains a 0.

## **\$\$JOE**

### **JOB**

Specifies that the JQE JOE chain should be searched for the first held JOE. \$\$GETHDJ returns in register 1 the address of the first held JOE. If there is no held JOE, register 1 contains a 0.

### **JQE=**

Specifies the address of the first JQE in the chain to search for held data sets. This operand is required if QUEUE=JOB is specified.

### **PSO=**

Specifies whether (YES) or not (NO) JES2 is to check if the JOE is available to the process SYSOUT (PSO) processor. PSO= is ignored if you also specify QUEUE=HOLD.

## **Environment**

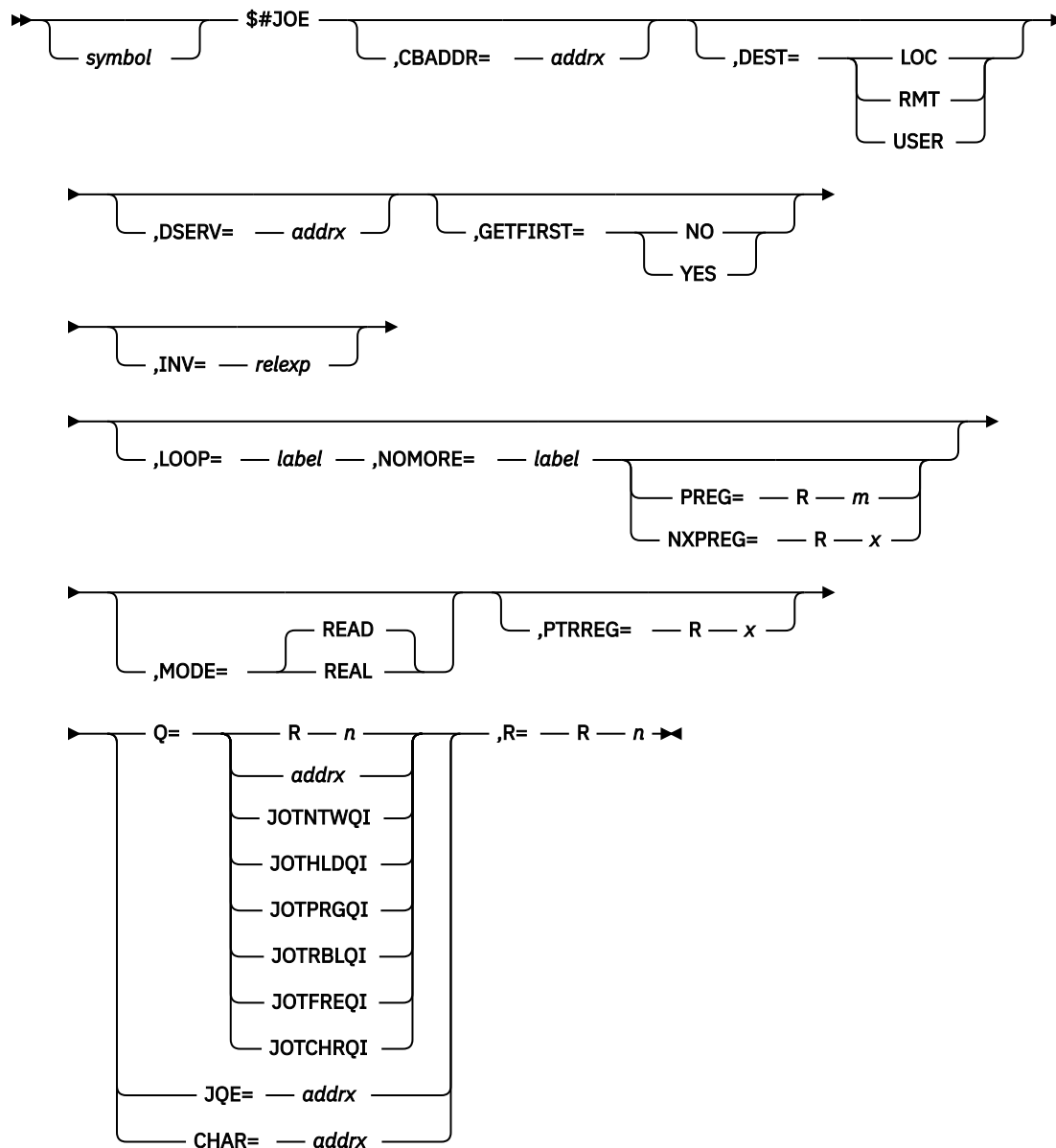
- JES2 main task.
- \$WAIT cannot occur.

## **\$\$JOE – Find and validate queue**

---

Use the \$\$JOE to cause the output service processors to generate the address for the head of a specified queue. You can then reference the first JOE on the queue through the JOENEXT field. You must establish addressability to the JOT before you use this macro instruction.

# Format description



## **CBADDR=addrx**

Specifies the address of JOA to use for processing. If this parameter is specified, the *addrx* value is passed to the first \$DOGJOE macro as CBADDR, and KEEP is specified on subsequent calls. If a field or register is specified, the macro saves the JOA address value in that field or register. If the value in the field or register is zero (0), the \$DOGJOE macro will allocate a new JOA, if necessary. This parameter is only valid if MORE=READ is also specified.



**Attention:** Do NOT specify this parameter as a DSECT label.

## **DEST=**

Specifies the destination queue within the class specified by the Q=operand. Possible values are as follows:

### **LOC**

The local queue for this class.

**RMT**

The remote queue for this class. **Do not code DEST=RMT if you specified Q=JOTNTWKQ or Q=JOTHLDQ.**

**USER**

The queue for all users in this class. **Do not code DEST=USER if you specified Q=JOTNTWKQ or Q=JOTHLDQ.**

**DSERV=addrx**

Specifies the address of a DSERV control block that is associated with a checkpoint version. If not specified, the real checkpoint is used. AR ASC mode must be turned on in order to specify DSERV=.

DSERV= is required if not running in the main task and **not** allowed if running in the main task.

**Note:** You must obtain the IAZDSERV data area, which describes the checkpoint version, prior to invoking the \$\$JOE service. Use a \$DSERV GET call to do so. See [Appendix D, "Accessing checkpoint control blocks outside the JES2 main task," on page 481](#) for a typical coding example.

**GETFIRST**

Indicates whether (YES), the default if JQE= is specified, or not (NO) the first JOE is to be obtained, rather than the zeroth JOE.

**Attention:**

- GETFIRST= is NOT valid if you also specify LOOP=.
- GETFIRST=NO is only allowed in conjunction with the Q= keyword or the CHAR= keyword and then only when the LOOP= keyword is not specified.

**INV=**

Specifies the label of the statement to which control is to be returned if the requested queue is invalid. If you omit this parameter, no check is made to ensure the validity of the queue. **Do not code this operand if you also specify Q=JOTNTWKQ or Q=JOTHLDQ.**

**LOOP=**

Specifies the name of a label the macro generates to get the next JOE.

**Attention:**

- When LOOP= and NOMORE= are specified, the macro generates the code needed to loop through all the JOEs on a queue.
- When looping back to loop, the value in R= must be unchanged from the value previously returned from the macro.
- LOOP= is optional, however, if you code LOOP= or NOMORE=, then both must be coded.

**NOMORE=**

Specifies the label of where to branch when there are no more JOEs on the queue.



**Attention:** NOMORE= is optional, however, if LOOP= or NOMORE= is coded, then both must be coded.

**MODE=REAL|READ**

Indicates whether the macro returns real JOEs or artificial JOEs (JOAs) that are in READ mode.

**REAL**

The macro \$\$JOE returns real JOEs.

**READ**

The macro \$\$JOE returns artificial JOAs that are in READ mode. READ is the default value of MODE.

**NXPREG=**

Specifies an optional work register used for prefetch processing when LOOP= is specified. This register should remain unchanged for the duration of the loop. When specified and running on a processor that support prefetch instruction, the next JOE in the chain is pre-fetched for read access when the macro returns the current JOE address. The NXPREG and PREG parameters are mutually exclusive.



**PTRREG=**

Specifies the register containing the index of the JOE returned in the R= parameter. The PTRREG parameter is valid only when MODE=REAL is specified and LOOP or GETFIRST=YES must be specified.

**PREG=**

Specifies the register containing the address of the JOE that pointed to the JOE returned in R=. This can be the 0th JOE. This is used if the JOE that was returned is placed on a different queue and you want to continue looping where you left off.

Place the value from this register into the register specified in R= to resume scanning the queues. This operand is optional and is only valid when MODE=REAL and LOOP= are specified. Valid registers are R2-R10 and R12. The PREG= parameter can be specified only in the JES2 main task.

**Q=**

Specifies the address or a register containing the address of the storage location containing the requested SYSOUT class or the address of the offset into one of the special queues. The possible queues that can be specified are:

**JOTNTWQI**

The network queue.

**JOTHLDQI**

The held queue.

**JOTPRGQI**

The purge queue.

**JOTRBLQI**

The rebuild queue.

**JOTFREQI**

The free queue.

**JOTCHRQI**

The characteristics JOE queue.

**JQE=**

Specifies the address of the JOE whose JOEs are to be processed. JOE=, Q=, or CHAR= must be specified.

**CHAR=**

Specifies the address of the CHAR JOE whose JOEs are to be processed. JOE=, Q=, or CHAR= must be specified.

**R=**

Specifies the register (Rn) into which the address of the desired queue head is to be loaded.

## Environment

- Main task.
- USER and SUBTASK environments.
- \$WAIT cannot occur.

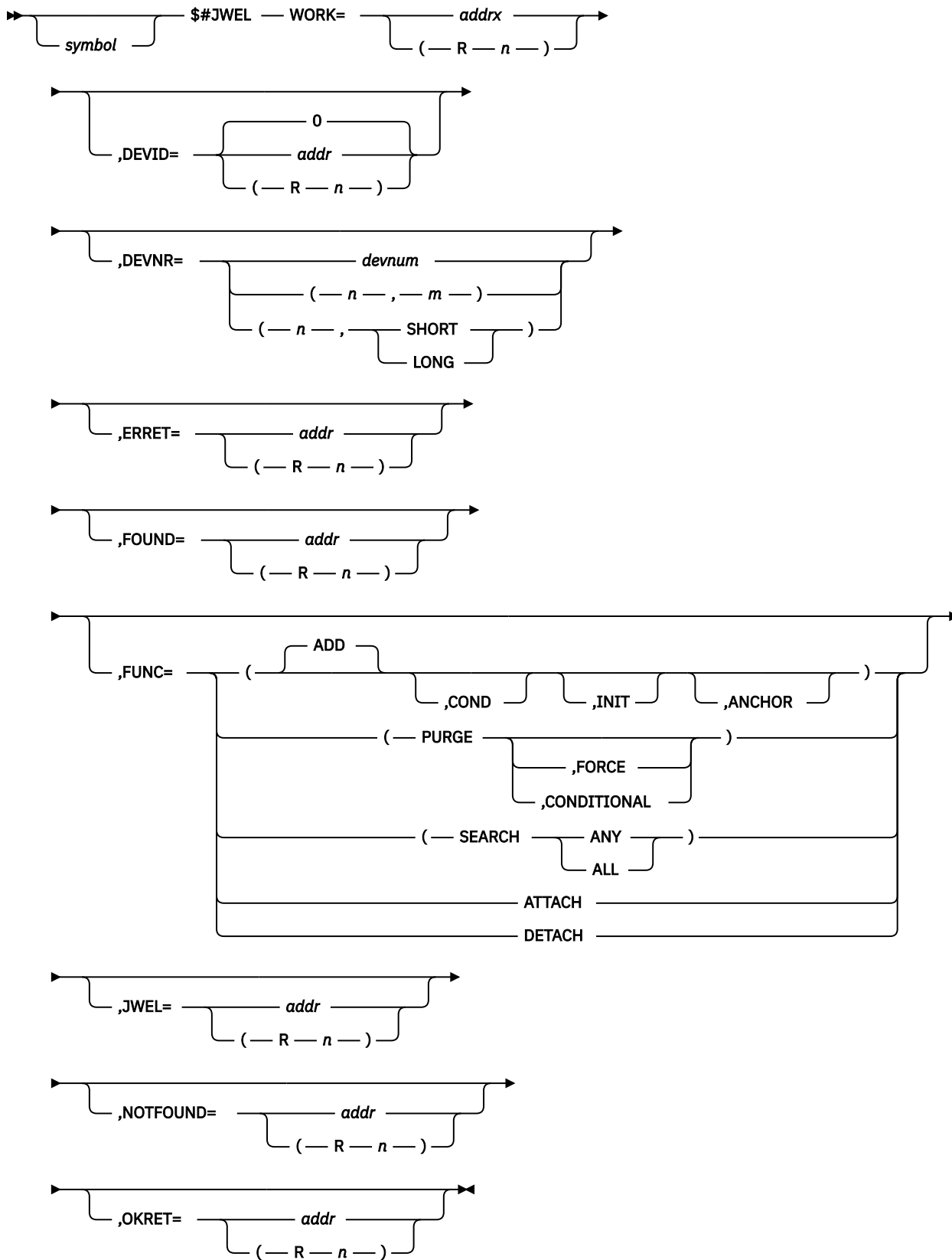
## \$\$JWEL – JOE writer exclude list (JWEL) services

---

Use \$\$JWEL to add, purge, attach, or detach (without release) a JOE writer exclude list (JWEL) or search for a JWEL already queued for a specific device.

The function for which you invoke this macro, dictates the required, optional, and unallowable parameters you can code on a particular call. Be certain to review [Table 3 on page 39](#) for a summary of parameter restrictions.

## Format description



**WORK=**

Specifies the address or a register (R2-R10) that contains the address of a JOE in the JOT whose JWELs require processing. An address of a JOA or a register (R2-R10) that contains the address of a JOA might also be specified.

**DEVID=**

Specifies the address or a register (R2-R10) that contains the address of a device ID associated with the JWEL. DEVID= can only be specified on a FUNC=ADD call. If not specified, JES2 sets this address to 0.

**DEVNR=**

For FUNC=ADD and FUNC=SEARCH calls, this parameter specifies the address or a register (R2-R10) that contains the address of a device number associated with the JWEL based on the type of function you request as follows:

**For FUNC=ADD**

Indicates a device number in one of the following formats:

- DEVNR=(*n*,*m*)

where:

- *n* is the address of a 4-byte number
- *m* is the address of an 8-byte number

JES2 uses only one of these values in the JWEL based on the following:

If *n* is 0, not supplied here, or points to 4 bytes of zeros, then JES2 uses *m* and assumes it to be a valid 8-byte number, otherwise JES2 uses the *n* specification.

- DEVNR=(*n*, SHORT | LONG)

where:

- *n* is the device address
- *SHORT* is a "short" device number
- *LONG* is a "long" device number

- DEVNR=*n*

where:

- *n* is an address or the complement of an address. If *n* is positive, then DEVNR= is the address of a 4-byte device number. If *n* is negative, then DEVNR= is the complemented address of an 8-byte device number.

**For FUNC=SEARCH**

Indicates a device number.

- DEVNR=(*n*,*m*)

where:

- *n* is the address of a 4-byte number
- *m* is the address of an 8-byte number

JES2 uses both values (*n* and *m*) if provided. *n* specifies the short device number, and *m* specifies the long device number. If either value is missing or point to a field of zeros, then JES2 suppresses a search for that length device number.

If *n* is 0, not supplied here, or points to four bytes of zeros, then JES2 uses *m* and assumes it to be a valid 8-byte number.

**ERRET=**

Specifies the label or register that contains the address of an error routine that is to receive control if processing is not successful (return code is non-zero).

**FOUND=**

Specifies the label or register that contains the address to which to branch if JES2 does find the JWEL on a FUNC=SEARCH call.

**FUNC=**

Specifies the function of this \$#JWEL call as follows:

See [Table 3 on page 39](#) for a summary of parameter restrictions associated with specific FUNC= calls.

**ADD[,COND]**

Indicates a request to JES2 to add a JWEL to the queue of JWELs for the specified JOE. COND indicates that JES2 should not add the JWEL if another JWEL with the same device number already exists.

**PURGE[,FORCE|CONDITIONAL]**

Indicates that JES2 should remove all JWELs associated with the specified JOE. FORCE

**,FORCE**

Indicates that JES2 is to unconditionally force purge processing.

**,CONDITIONAL**

Indicates that 4 byte JWELs, created for SAF reasons, will be eliminated. The 8 byte JWELs, created for SAPI reasons, will be eliminated only if the JOE and the JWEL no longer match in their timestamp or the JOE is on the free queue. If JWELs are removed \$#TJEV will be called to delete the TJEV elements.

**Note:** INIT places the JOE creation time in the JWEL anchor.

**,INIT**

Places the JOE creation time in the JWEL anchor.

**,ANCHOR**

Determines the address of the JWEL anchor for the given JOE.

**SEARCH**

Indicates that JES2 is to check if a JWEL with the same device number already exists.

**ANY**

Optionally, indicates that JES2 should return an return code of 4 if JES2 finds any non-transient (non-\$JWEBULK) JWELs for the JOE.

**ALL**

Optionally, indicated that the JWELs that JES2 finds must match the search criteria. If any JWEL does not match, or JES2 finds no JWELs, JES2 returns a return code of 0. JES2 does not consider transient (\$JWEBULK) JWELs in this search.

**Note:** If you do not include either ANY or ALL, then JES2 searches for a match on either form (n or m) of the DEVNR= parameter and returns a return code of 4 if found.

**ATTACH**

Indicates that JES2 is to attach a chain of JWELs to the JOE.

**DETACH**

Indicates that JES2 is to remove all JWELs from the JWEL anchor without freeing them. Use this prior to a subsequent ATTACH call.

**JWEL=**

Specifies the label or register that contains the address of the first JWEL in the chain of JWELs that JES2 is to attach to a specified JOE.

You can only specify JWEL= on a FUNC=ATTACH call.

**NOTFOUND=**

Specifies the label or register that contains the address to which to branch if JES2 does not find the JWEL on a FUNC=SEARCH call.

### OKRET=

Specifies the label or register that contains the address of a routine that is to receive control if processing is successful (return code is 0).

## Programming considerations

Based on the function (FUNC=) for which you are requesting this call, the set of required, optional, and unallowable parameters can be summarized as follows:

Table 3. Summary of ##JWEL Parameter Requirements and Restrictions						
Parameters	ADD	SEARCH	PURGE	ATTACH/ DETACH	INIT	ANCHOR
WORK=	Required	Required	Required	Required	Required	Required
DEVID=	Optional	Invalid	Invalid	Invalid	Invalid	Invalid
DEVNR=	Required	Required	Invalid	Invalid	Invalid	Invalid
ERRET=	Optional	Invalid	Optional	Invalid	Invalid	Invalid
FOUND=	Invalid	Optional	Invalid	Invalid	Invalid	Invalid
JWEL=	Invalid	Invalid	Invalid	*	Invalid	Invalid
NOTFOUND=	Invalid	Optional	Invalid	Invalid	Invalid	Invalid
OKRET=	Optional	Invalid	Optional	Invalid	Invalid	Invalid

\* Required for ATTACH, Invalid for DETACH.

## Return codes

The following return codes (decimal) are returned in register 15.

### Return Code Meaning

#### 0

- ADD - element added
- ATTACH - only return code provided
- DETACH - only return code provided
- INIT - only return code provided
- ANCHOR - only return code provided
- PURGE - JWEL chain emptied
- (PURGE,CONDITIONAL) - JWEL chain is now empty
- SEARCH - device with same number not found
- (SEARCH,ALL) - no JWELs found or not all JWELs match the search criteria
- (SEARCH,ANY) - no JWELs found

#### 4

- ADD - element not added
- PURGE - JWEL chain left unchanged
- (PURGE,CONDITIONAL) - JWEL chain left intact
- SEARCH - device with same number found
- (SEARCH,ALL) - at least one JWEL found; all JWELs found match the search criteria
- (SEARCH,ANY) - one or more JWELs exist for this JOE

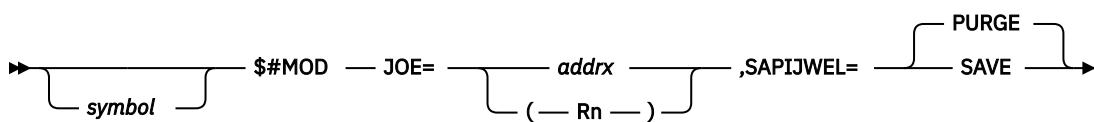
## 8

- ## Environment

- Main task.
- \$WAIT cannot occur.

Use `$$MOD` to remove a work JOE from the queue it is currently on and to place it on the proper queue as determined by its routing (JOEROUT) or SYSOUT class (JOECURCL). `$$MOD` should be issued after a JOE's queue status has been changed.

## Format description



Specifies the address of the work JOE or the JOA that contains the work JOE that is moved from one queue to another.

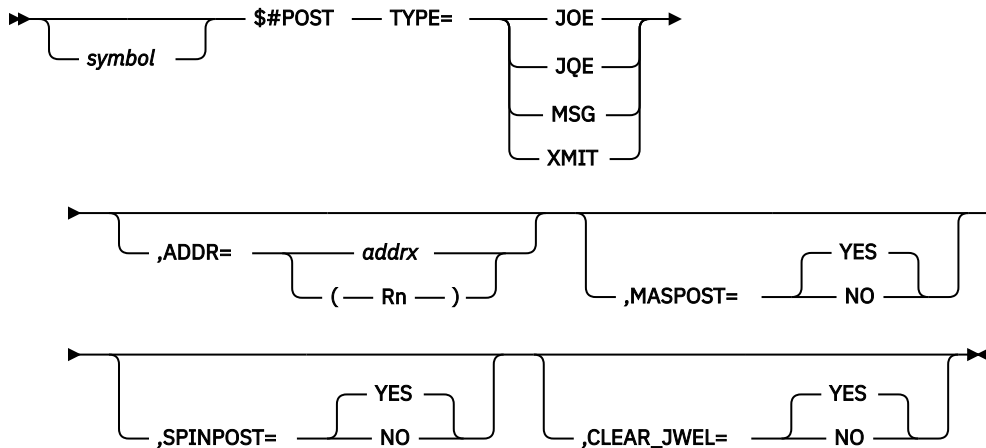
Specify SAVE or PURGE. SAVE means that only the SAF JWELs are purged in this operation. PURGE means that both SAF and SAPI JWELs are purged.

## Environment

- Main task.
- \$WAIT can occur.

Use `$$POST` to post device processors that are waiting for work associated with specific output devices. `$$POST` ensures that when a new piece of work becomes available for processing, only those processors associated with the devices eligible to select the work are posted. JES2 uses `$$POST` when a JOE is added or returned to the JOT. `$$POST` is also used 1) when a message is spooled for a remote, 2) when a node's remote or local output device becomes available for use, 3) when a console or printer is added to notify when a node, remote processor, or device becomes available for use, or 4) when a new path becomes available to a node.

## Format description



### TYPE=

Specifies what type of ##POST to issue. You can specify one of four types. JOE specifies a work JOE ##POST. JQE specifies a JQE and associated work JOEs ##POST. MSG specifies a spooled message ##POST. XMIT specifies a SYSOUT transmitter ##POST. You must specify this operand; there is no default.

### ADDR=

Specifies an address. The address depends on the TYPE selected. For TYPE=JOE, ADDR is the address of the work JOE that is to be ##POSTed. For TYPE=JQE, ADDR is the address of the JQE whose work JOEs are to be ##POSTed. For TYPE=MSG, ADDR is the address of the route code for the remote printers or consoles that are to be ##POSTed. For TYPE=XMIT, ADDR is the address of the line DCT associated with the SYSOUT transmitters that are to be ##POSTed; if this address is specified as zero, then all SYSOUT transmitters that are waiting are ##POSTed.

### MASPOST=

Specifies whether the work JOEs that are to be ##POSTed should have their JOE post flags reset so that the post is propagated to all members in a multi-access spool complex. MASPOST= is valid only when TYPE=JOE or TYPE=JQE is specified.

#### Note:

1. The MASPOST flag is passed in the first byte of the inline parameter list.
2. You need control of the checkpoint data set (obtained through \$QSUSE) before issuing this macro.

### SPINPOST=

Specifies whether POST is to be done for SPIN JOEs. SPINPOST= is valid only when TYPE=JQE is specified.

### CLEAR\_JWEL=

Specifies whether the JWELs associated with the JOE are to be cleared (deleted). CLEAR\_JWEL=YES is valid for TYPE=JOE only. The default is CLEAR\_JWEL=YES.

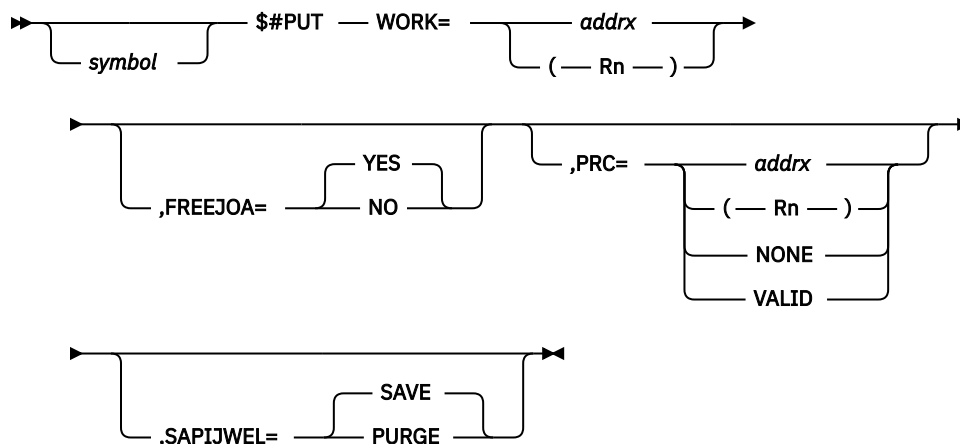
## Environment

- Main task.
- \$WAIT cannot occur.

## ##PUT – Return an unfinished job output element (JOE) to the JOT for later processing

Use ##PUT in a processor to return a JOE to the JOT for later processing. Optionally, the status of the JOE is maintained for a warm start of the system or restart of the work.

## Format description



### WORK=

Specifies the address of a work JOE or JOA containing a work JOE that is to be returned to the JOT class queues for future selection.

### FREEJOA=

Specifies optional actions for the \$#PUT service if this macro is called with an update mode JOA. YES is the default.

#### YES

Indicates that the following actions should be done:

- Write the JOA changes to checkpoint.
- Free the JOA of the caller and all user stack elements.
- Release the BERT lock.

#### NO

Indicates that before returning to caller, changes are written to checkpoint but the JOA is not freed.

### PRC=

Specifies the address of a checkpoint buffer if the current status of the work item is to be stored. If PRC= is not specified or is specified as PRC=NONE, the work item is reset to reflect its initial entry status. If PRC=VALID is specified, no change is made to the current status of the work item.

### SAPIJWEL=

Specify SAVE or PURGE. SAVE means that only the SAF JWELs are purged in this operation. PURGE means that both SAF and SAPI JWELs are purged.

## Environment

- Main task.
- \$WAIT can occur.

## **\$#REM – Remove a work/characteristics JOE pair from the JOT**

Use \$#REM to remove a work and characteristics JOE pair from the JOT after the output requirement they represent has been satisfied.

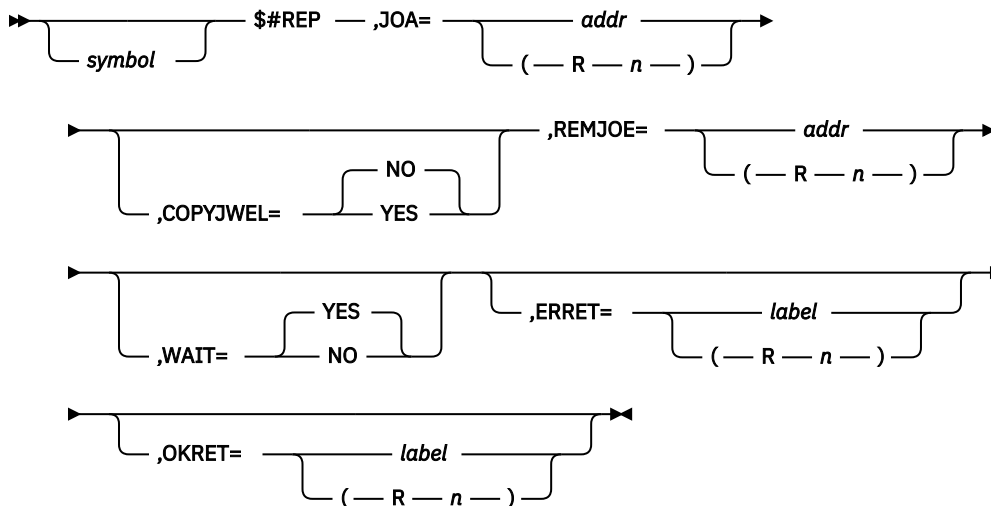




## ##REP – Replace a work or characteristics JOE

Use `$$REP` to replace a work JOE with a new work and characteristics JOE. JES2 checks to be certain that enough free JOEs are available to add the new JOE. If JES2 determines there are not enough JOEs, it optionally issues a `$WAIT` on behalf of the caller and creates the required free JOEs.

## Format description



**JOA=**

Specifies the address or a register (R2-R10) that contains the address of a prototype or real artificial JOE (JOA) whose content replaces the current real work JOE and characteristics JOE content.

**Note:**

1. A real JOA must be specified if REMJOE= is not specified.
2. If a read mode JOA is passed on JOA=, this service automatically refreshes the JOA before returning to its caller.

## COPYJWEL=

Specifies that the JWELs associated with the JOE that is to be replaced are to be copied to the JOE being created (YES) or discarded (NO).

**REMJOE=**

Specifies the address or a register (R2-R10) that contains the address of the work JOE that is replaced by this service.

**Note:** REMJOE= must be specified if a prototype JOA is specified for the JOA= parameter.

**WAIT=**

Specifies whether (YES) or not (NO) JES2 is to issue a \$WAIT macro if the JOT is full and cannot immediately add the new JOE. WAIT=YES is the default but if overridden with WAIT=NO and the JOT is also full, processing fails with return code 4.

**ERRET=**

Specifies a label or register that contains the address of a routine to receive control if processing is not successful (a non-zero return code is returned in R15).

**OKRET=**

Specifies a label or register that contains the address of a routine to receive control if processing is successful (a 0 return code is returned in R15).

## Return codes

The following return codes (decimal) are returned in register 15.

**O**

Processing successful. New JOE added.

4

Processing failed. The JOT is full; JES2 could not issue a \$WAIT to create a new free JOE because WAIT=NO was explicitly specified.

8

Processing failed. The SYSOUT class was not valid; therefore, JES2 did not remove the JOE specified.

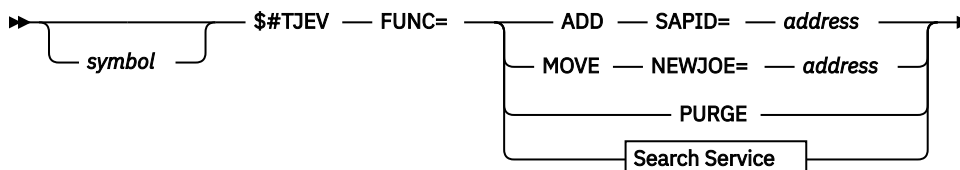
## Environment

- Main task.
- \$WAIT can occur.

## \$#TJEV – Manage the thread JOE exclusion vector

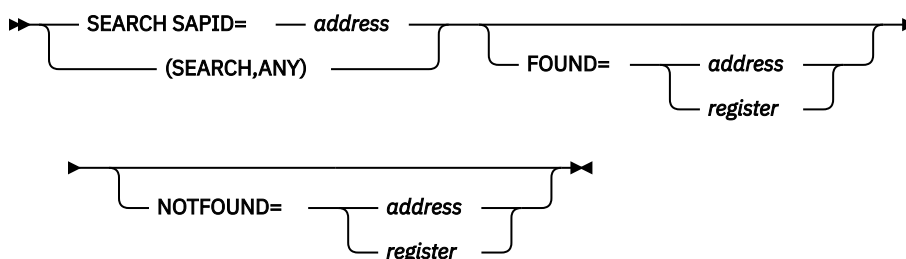
Use `$_TJEV` to manage the thread JOE exclusion vector. There are at most one of these vectors for each SAPIID. The vector provides a means for an application to state that a given data set should be kept, but never shown to the keeping thread again.

## Format description



► JOE= — *address* ◄

## Search Service



**FUNC=**

Specifies the service being requested.

**ADD**

Use the JOE address and SAPID address provided to turn on the bit representing the given JOE in the TJEV pointed to by the SAPID. If no TJEV exists for the SAPID, create one and turn on the bit.

## SEARCH

Use the JOE and SAPID addresses provided to see if the JOE is excluded from the selection by the thread. FOUND= and NOTFOUND= are used to branch to the appropriate logic point.

## SEARCH.ANY

Use the JOE address provided and examine all TJEVs in the system to see if any of them has the bit turned on for the JOE provided. FOUND= and NOTFOUND= are used to branch to the appropriate logic point.

## \$ACTIVE

### **PURGE**

Use the JOE address provided and examine all TJEVs in the system. Ensure all TJEVs have the bit turned off that represents the JOE.

### **MOVE**

Use the JOE= and NEWJOE= addresses provided to move the TJEV setting for the JOE to the setting for the NEWJOE. All TJEVs are altered. When finished, the bit corresponding to the JOE is turned off in every TJEV.

### **JOE=**

Specifies the address of the work JOE or JOA whose corresponding bit in the TJEV is being managed.

### **SAPID=**

Specifies the address in the SAPID data space of the control block representing the thread.

### **Note:**

1. Invalid operand if FUNC=PURGE, MOVE, or SEARCH,ANY.
2. Required operand if FUNC=ADD or SEARCH.

### **NEWJOE=**

Specifies the address of the JOE or JOA whose setting should be moved from the JOE indicated by the JOE= parameter.

**Note:** Valid only if FUNC=MOVE.

### **FOUND=**

Specifies a label to be branched to or a register to be branched on if the bit is found when FUNC=SEARCH is specified.

### **NOTFOUND=**

Specifies a label to be branched to or a register to be branched on if the bit is not found when FUNC=SEARCH is specified.

## Return codes

The following return codes (decimal) are returned in register 15.

### **Return Code**

#### **Meaning**

**0**

The return code is always zero for FUNC=ADD,PURGE,MOVE.

For the two SEARCH functions, 0 means not found.

**4**

For the two SEARCH functions, 4 means that the bit representing the JOE was found.

**8**

For the SEARCH,ANY function, 8 means that the bit representing JOE was found only in TJEV pointed to by the given SAPID.

## Environment

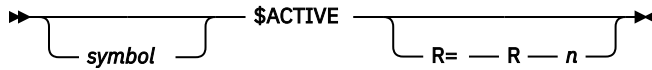
JES2 main task

## \$ACTIVE – Specify processor is active

---

Use \$ACTIVE to indicate to JES2 that the associated processor is performing activities for the JES2 main task; this prevents JES2 from being cleanly withdrawn from the system (through \$P JES2) when JES2 is processing a job or task.

## Format description



### R=

Specifies the work register which is to be used by the \$ACTIVE macro instruction. Do not enclose the register (R=) value in parenthesis. Register 1 is the default.

### Note:

1. JES2 is considered active when the active count is greater than 0 (\$DORMANT decreases the active count). When the active count is 0, JES2 issues \$HASP099.
2. Do not use R=0.

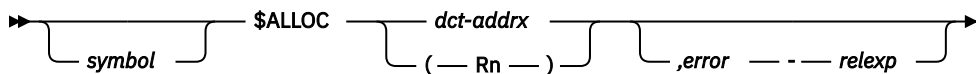
## Environment

- Main task.
- \$WAIT cannot occur.

## \$ALLOC – Allocate a unit record device

Use \$ALLOC to allocate a unit record or teleprocessing device to JES2.

## Format description



### dct

Specifies the address of the DCT to be allocated.

### error

Specifies a location to which control is returned if the device (DCT) cannot be allocated. The condition code is set to reflect the allocation of the DCT as follows:

#### CC=0

The device could not be allocated.

#### CC≠0

The device was successfully allocated.

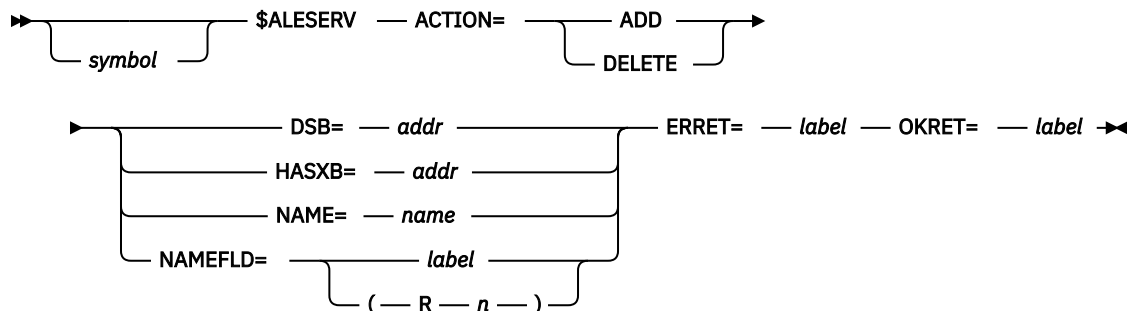
## Environment

- Main task.
- \$WAIT can occur.

## \$ALESERV – JES2 ALET services

Use \$ALESERV to add or delete access list entry tables (ALETs). This macro service also maintains a list of the ALETs for JES2-owned address spaces.

## Format description



### **ACTION=**

The action to be taken:

#### **ADD | DELETE**

Specifies that JES2 is to add (ADD) or delete (DELETE) an ALET.

**Note:** You must also specify one and only one of the following: DSB=, HASXB=, NAME=, or NAMEFLD=.

### **DSB=**

Specifies the address of the data space block (DSB) associated with the data space for which you want an ALET added or deleted.

If you specify DSB=, you cannot specify HASXB=, NAME=, or NAMEFLD=.

### **HASXB=**

Specifies the address of the address space extension block (HASXB) for which a *DELETE ALL* is required. HASXB= implies NAME=ALL and can only be specified on a ACTION=DELETE call.

If you specify HASXB=, you cannot specify DSB=, NAME=, or NAMEFLD=.

### **NAME=**

Specifies the name of the ALET requested. This name must match either the name specified on a \$DSPSERV CREATE call or one of the predefined names in \$ALINDEX. NAME=ALL requests that JES2 delete all ALETs. NAME=ALL can only be specified on a ACTION=DELETE call.

If you specify NAME=, you cannot specify DSB=, HASXB=, or NAMEFLD=.

### **NAMEFLD=**

Specifies a label that contains the name of the ALET or a register that points to the name of the ALET to be processed. This name must match either the name specified on a \$DSPSERV CREATE call or one of the predefined names in \$ALINDEX. A field that contains the value *ALL* is equivalent to specifying NAME=ALL and only allowable on a ACTION=DELETE call.

If you specify NAMEFLD=, you cannot specify DSB=, HASXB=, or NAME=.

### **ERRET**

The label to branch to in the event of a non-zero return code.

### **OKRET**

The label to branch to if the return code is zero.

## Programming notes

- If JES2 has already added a requested ALET, a new ALET is not added.
- If the ALINDEX does not exist, JES2 creates it on the first ADD call.
- If this is a DELETE call and NAME=ALL (or implied), JES2 deletes the ALINDEX table.
- If all ALETs in the ALINDEX table are deleted, the table is **not** also deleted.

## Return codes

The following return codes (decimal) are returned in register 15.

### Return Code Meaning

<b>0</b>	Processing successful. If ADD call, ALET is returned in R1.
<b>4</b>	Processing failed. Zero returned in R1.
<b>8</b>	Processing failed. JES2 cannot identify the ALET identifier passed to it. Zero returned in R1.
<b>12</b>	A logic error occurred due to a mismatch of the action requested (ADD or DELETE) and the ALET identifier passed to JES2, such as \$ALESERV ADD,NAME=ALL.

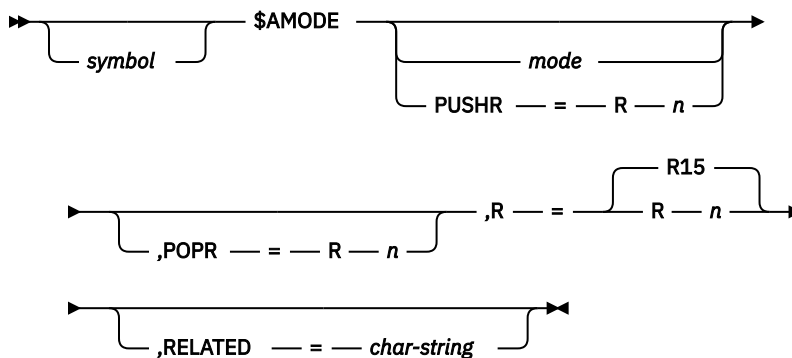
## Environment

- JES2 main task, subtask, functional subsystem (HASPFSM), or user environment.
- \$WAIT cannot occur.
- Callers can be in AR ASC mode.

## \$AMODE – Set the addressing mode

Use the \$AMODE macro instruction to set 24-bit and 31-bit addressing modes.

## Format description



### mode

Specifies the addressing mode to be used by the code that follows this macro until it is again specified. This is a positional parameter and must be specified if PUSHHR= is also specified. Do not use this operand if POPR= is specified.

#### 24

Specifies 24-bit addressing mode.

#### 31

Specifies 31-bit addressing mode.

### PUSHHR=

Specifies a register to be used to store the current addressing mode. If mode is specified, this keyword is also required.

**Note:** Do not enclose the specified register in parenthesis.

## \$ARMODE

### POPR=

Specifies a register to be used to restore the previous addressing mode. The register specified here must have been previously loaded by a \$AMODE mode PUSHHR= instruction. Do not specify this keyword if mode and PUSHHR= are specified.

**Note:** Do not enclose the specified register in parenthesis.

### R=

Specifies a work register to be used by this macro instruction. Register 15 is the default.

**Note:** Do not enclose the specified register in parenthesis.

### RELATED=

Specifies a character string used to self-document this macro instruction call. Any specification type value for macro keywords can be used here. This field is useful for documenting the inline pairing of \$AMODE macros.

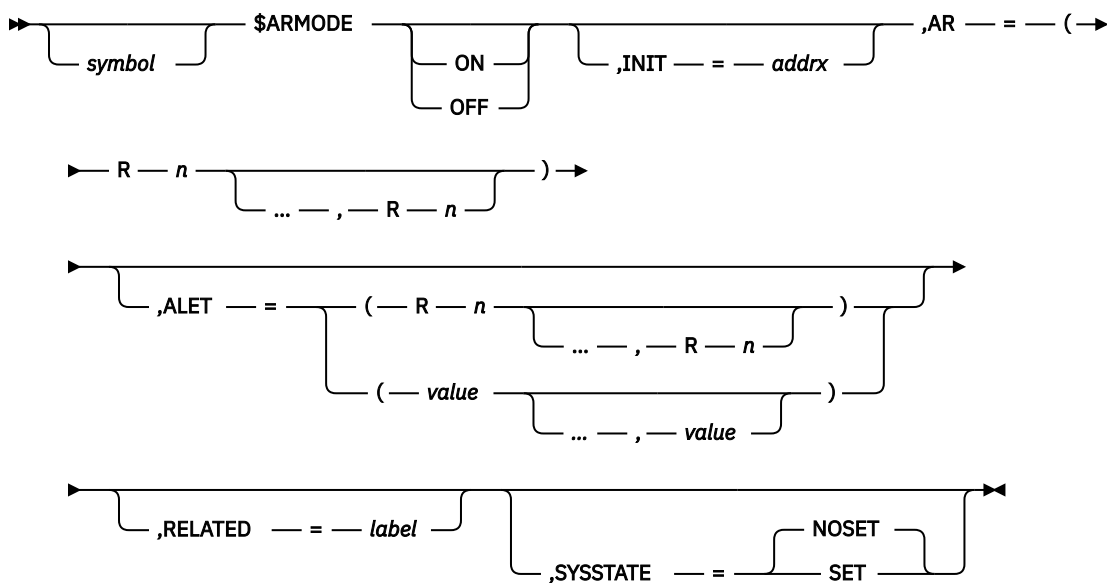
## Environment

- JES2 main task, JES2 subtask, FSS, and user environment.
- Waits cannot occur.

## \$ARMODE – JES2 multi-address space access

Use the \$ARMODE macro instruction to perform functions related to multi-address space management, such as switching access register control mode, and how the access registers are to be loaded.

## Format description



### ON|OFF

Specifies whether the call to \$ARMODE is to set the access register control mode ON or OFF. If you do not set this value, the current AR-mode remains in effect.

**Note:** If this parameter is specified it **must be coded first**.

### INIT=

Specifies that all access registers are loaded with the 16 consecutive full-words beginning at the storage location pointed to by this address.



**AR=**

Specifies the access registers that are loaded with the values specified by ALET=. These registers are set in the order specified. You can specify up to 16 access registers.

**ALET=**

Specifies a list of values (ALETs) that are set in the access registers specified by AR=. You can specify either one or more registers or one or more fullword values. If any or all of these ALET values are not specified, the corresponding registers are set to 0. Each ALET points to an entry on an access list. For more information, see [“Data space usage”](#) on page 479.

**RELATED=**

Specifies a valid alphanumeric label that this macro is related to.

**Note:** If the label referred to by this keyword does not exist, an assembler error will occur.

**SYSSTATE=**

Specifies whether (SET) or not (NOSET) a SYSSTATE macro is issued. The SYSSTATE macro is used to indicate that a caller is in AR mode and provides the generation of code and addresses that are appropriate for callers in that mode.

**Note:**

1. The contents of general registers 0 and 1 are destroyed across a call to \$ARMODE.
2. If ALET= is specified, AR= must be specified.

## Environment

- All JES2 environments.
- MVS WAITs or JES2 \$WAITs cannot occur.

## \$BERTTAB – Map block extension reuse table (BERT) table entries

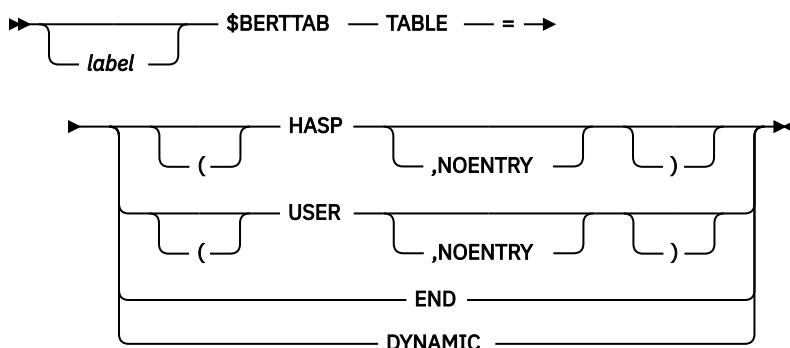
Use \$BERTTAB to map and generate BERT table entries. BERTTAB entries are used to define data to be stored in the BERT portion of the checkpoint. \$BERTTAB is used to define the start or end of a table, entries within the table, or search keys.

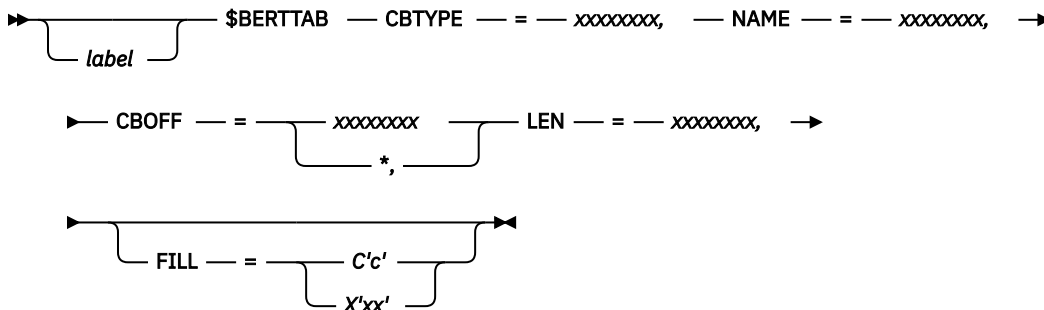
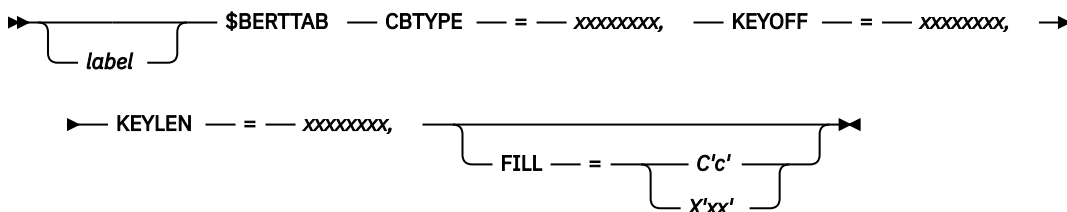
## Format description

**Note:** The format description that follows breaks the macro into three sections:

- Boundary form – the form that starts or ends a table.
- Data definition form – the form that defines specific data to be placed in the BERT.
- Search key form – the form that defines how to locate a specific element.

### Boundary form



**Data definition form****Search key form****TABLE=**

Specifies the start (TABLE=HASP|USER|DYNAMIC) and end (TABLE=END) of a BERT table.

**TABLE=HASP**

Specifies that this is a HASP table.

**TABLE=USER**

Specifies that this is a USER table.

**TABLE=DYNAMIC**

Specifies that this is a DYNAMIC table.

**TABLE=END**

Specifies that this is the end of the BERT table.

**Note:** If TABLE= is specified, all other operands are ignored.

**CBTYPE=**

Specifies the type of control block represented by this table entry. This may be specified as JQE, CAT, WSCQ, or a 1-8 character user-defined type. Non-IBM types should start with either a U or a V to avoid conflict with future IBM types.

**NAME=**

Specifies a 1-8 character name that represents the specific data within the control block type (CBTYPE). The name must be unique within a control block type. USER and DYNAMIC table entries may not use the same name as HASP entries within the same CBTYPE. Non-IBM types should start with either a U or a V to avoid conflict with future IBM types.

**CBOFF=**

Specifies the offset into the control block to which data from this entry should be moved. CBOFF=\* may be specified to indicate that the data should be placed at the highest unused offset. To retrieve the offset and length that was associated with this data at run time, use \$DOGBERT ACTION=GETOFFSET, with CBTYPE and NAME equal to CBTYPE and NAME from the table entry.

**LEN=**

Specifies the length of the data.

**KEYOFF=**

Specifies the offset of the search key for this control block type.

**Note:** Only one search key entry may be specified for any control block type.

**KEYLEN=**

Specifies the length of the search key.

**FILL=**

Specifies the fill character for this data area. This parameter is optional, and the default is X'00' for data entries and C' ' for search key entries.

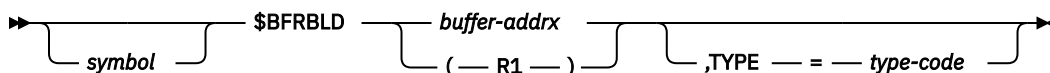
## Environment

- JES2 main task or initialization.
- \$WAIT is not applicable; this macro defines a static table.

## \$BFRBLD – Construct a JES2 buffer prefix

Use \$BFRBLD to construct an IOB or RPL in the front of a JES2 buffer. The IOB or RPL is used to read into or write from the data portion of the buffer.

## Format description

**buffer**

Specifies the address of a buffer where the prefix (an IOB or an RPL) is to be constructed. If an address is used, it specifies a word in storage containing the buffer address.

If a register notation is used, the buffer address must have been loaded into the designated register before the execution of this macro instruction.

**TYPE=**

Identifies the type of buffer and specifies whether an IOB or RPL is to be constructed at the beginning of the buffer, according to the type code as follows:

**HASP (default)**

A local buffer; an IOB is to be constructed

**BSC**

A TP buffer; an IOB is to be constructed

**PAGE**

A local 4096-byte buffer; an IOB is to be constructed

**PP**

A local print/punch buffer; an IOB is to be constructed

## Environment

- Main task.
- \$WAIT cannot occur.

## \$BLDMSG – Build a message line

JES2 provides a message building facility to allow you to replace an existing \$HASPnnnn message or add a new \$HASPnnnn message. Use the \$BLDMSG macro to build and generate a message. Specifically JES2 reserves messages in the \$HASP9nn and \$HASP9nnn range for your use. See [z/OS JES2 Messages](#) for information on these reserved message ranges.

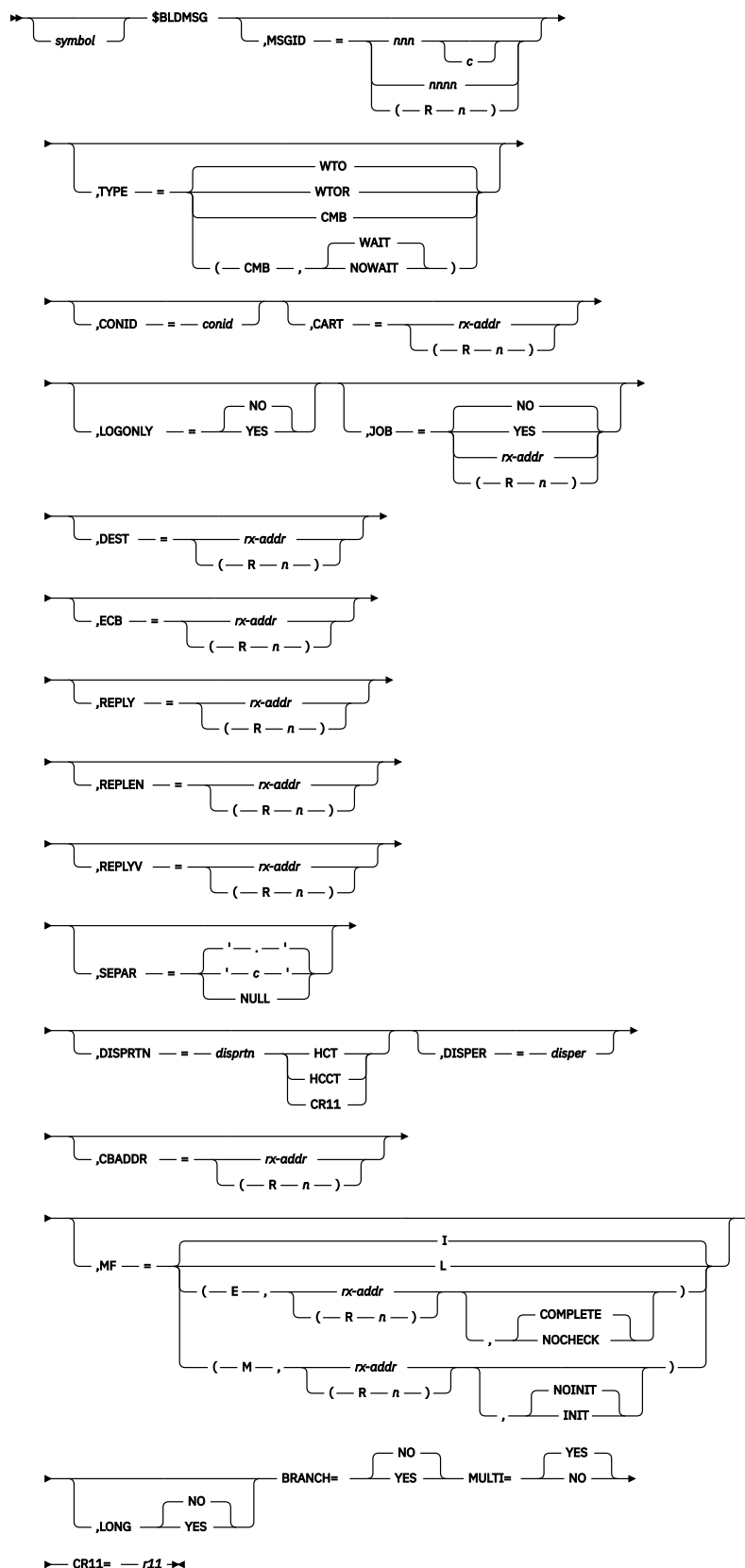
Use the \$SCANTAB macro to define a message to the message table. \$BLDMSG then builds the specific message using the \$SCAN facility, accessing the message table.

Parameters to be coded depend on the macro form used:

## **\$BLDMSG**

- MF=I (inline form)
- MF=L (list form)
- MF=E (execute form)
- MF=M (modify form).

## Format description



### MSGID=

Specifies a 3- or 4-digit JES2 message ID prefixed by \$HASP. Optionally, you can specify the fourth digit as a character suffix character to distinguish multiple definitions of the same message number

in the message table. You can specify these characters as the operand, for example MSGID=273A, or you can specify register notation.

If you specify register notation, it must contain the address of the message ID (left justified and padded with blanks if necessary). Register notation can be used for macro forms execute (E), inline (I), or modify (M). Register notation may not be specified on the list macro form (L).

This keyword is valid for all macro forms; it is required for inline or execute (with the COMPLETE parameter specified or as the default) macro forms. The last value that was set for this parameter will be used for the execute macro form with the NOCHECK parameter specified or the modify macro form with the NOINIT parameter specified (or as the default).

**TYPE=**

Specifies the type of message this routine builds and writes.

For inline and execute forms of the macro, WTO is the default type. Other macro forms do not have a default.

**WTO**

Indicates a write to operator (WTO) message. The message is issued from the JES2 main task. This is the default for the inline or execute (with the COMPLETE specified or as the default) macro forms. WTO can be specified on all macro forms. The last value that was set for this parameter will be used for the execute macro form with the NOCHECK parameter specified or the modify macro form with the NOINIT parameter specified (or as the default).

**WTOR**

Indicates a write to operator with response (WTOR) message. The message is issued from the JES2 main task. WTOR cannot be specified on the list macro form. If WTOR is specified, the ECB=, REPLY=, and REPLEN= parameters are also required on the same macro instruction.

**CMB**

Indicates the message is built in CMBs (console message buffers) and is passed to the \$WTO service for processing. CMB can be specified on all macro forms.

If the type CMB is specified, a second sub-operand, WAIT or NOWAIT, can also be specified.

**WAIT|NOWAIT**

Indicates to the display routine whether (WAIT) or not (NOWAIT) to \$WAIT if a CMB is not available. WAIT indicates that the display routine can \$WAIT. NOWAIT indicates that a CMB should be obtained even if a CMB is not available (when a processor cannot wait). For more information about obtaining a CMB, see [“\\$GETCMB – Get console message buffers” on page 193](#).

The default is WAIT.

**Note:**

1. If you code TYPE=CMB or TYPE=(CMB,WAIT), exit 10, when it receives control, is allowed to \$WAIT.
2. During JES2 initialization, the default \$BLDMSG display routine processes TYPE=CMB as TYPE=WTO.

**CONID=**

Specifies the 4-byte identifier of the console where the message is displayed. This parameter may be specified using:

- A label for a field containing the console ID.
- A base-displacement expression that is the address of the console ID.
- Register notation (R2 through R12) to specify a register containing the console ID.

It is valid for execute, inline, and modify macro forms. The default is to omit the console identifier for inline or execute (with the COMPLETE parameter specified or as the default) macro forms. The last value that was set for this parameter will be used for the execute macro form with the NOCHECK parameter specified or the modify macro form with the NOINIT parameter specified (or as the default).

**CART=**

Specifies the address of the command and response token to be used to issue messages. This can be specified as an rx-addr expression or using register notation (R2 through R12). It is valid for execute, inline, and modify macro forms. The default is to omit the token for inline or execute (with the COMPLETE parameter specified or as the default) macro forms. The last value that was set for this parameter will be used for the execute macro form with the NOCHECK parameter specified or the modify macro form with the NOINIT parameter specified (or as the default).

**LOGONLY=**

Specifies whether (YES) or not (NO) a message is only logged in the hardcopy log. If LOGONLY=NO is specified, the message appears on appropriate consoles and in the hardcopy log.

This keyword is valid for all macro forms when used with the TYPE=WTO|CMB parameter. The default is LOGONLY=NO for list, inline or execute (with the COMPLETE parameter specified or as the default) macro forms. The last value that was set for this parameter will be used for the execute macro form with the NOCHECK parameter specified or the modify macro form with the NOINIT parameter specified (or as the default).

**JOB=**

Specifies whether the message is preceded by a job identifier. Specify one of the following keywords:

**YES**

Specifies that field PCEJQE must contain the address of the JQE from which the job ID is built.

**NO**

Specifies that no job ID will be placed to the left of the message ID.

**rx-addr or (Rn)**

Specifies the address of the job ID. This can be an rx-addr address expression or register notation (R2-R12) containing the address of the eight character job ID.

This keyword is only valid for TYPE=CMB. JOB= is valid for all macro forms, however an address is not valid on list macro form. The default is JOB=NO for list, inline, and execute (with the COMPLETE parameter specified or as the default) macro forms. The last value that was set for this parameter will be used for the execute macro form with the NOCHECK parameter specified or the modify macro form with the NOINIT parameter specified (or as the default).

**DEST=**

For a remote destination, specifies the address of the symbolic destination for which a binary route code is obtained. This parameter can be specified using:

- Register notation (R2 thru R12) specifies a register containing the address of the symbolic destination.
- A label for a field containing the symbolic destination.

If register notation is used, the address of the symbolic destination must be loaded into the designated register prior to execution of the macro instruction.

This keyword may be specified for MF=E, MF=I or MF=M macro forms.

This keyword is only allowed if TYPE=CMB and MULTI=NO is specified.

The symbolic destination specified is not checked and any non-blank value specified is assumed to be a remote destination.

**ECB=**

Specifies the address of an ECB to be used for WTOR processing. This can be specified as an address expression or register notation (R2-R12).

This keyword is valid only if TYPE=WTOR is specified, and must be specified at the same time TYPE=WTOR is specified. It is valid for execute, inline, and modify macro forms. The last value that was specified for this parameter will be used for the execute macro form with the NOCHECK parameter specified or the modify macro form with the NOINIT parameter specified (or as the default).

**Note:** The ECB is set to zero before the MVS WTOR is issued.

**REPLY=**

Specifies the address of a response area for WTOR processing. This parameter may be specified as an rx-addr address expression or using register notation (R2 through R12).

This keyword is valid only if TYPE=WTOR is specified, and must be specified at the same time TYPE=WTOR is specified. It is valid for execute, inline, and modify macro forms. The last value that was specified for this parameter will be used for the execute macro form with the NOCHECK parameter specified or the modify macro form with the NOINIT parameter specified (or as the default).

**Note:** The reply area is set to blanks before the MVS WTOR is issued.

**REPLEN=**

Specifies the length of the reply area specified by the REPLY= parameter. This parameter may be specified as an rx-addr address expression or using register notation (R2 through R12).

This keyword is valid only if TYPE=WTOR is specified, and must be specified at the same time TYPE=WTOR is specified. It is valid for execute, inline, and modify macro forms. The last value that was set for this parameter will be used for the execute macro form with the NOCHECK parameter specified or the modify macro form with the NOINIT parameter specified (or as the default).

**REPLYV=**

Specifies the address of a reply vector (generated with a \$REPLYV macro instruction). The reply vector contains valid replies for this WTOR and an associated A-type address constant for each valid reply. The REPLYV= parameter may be specified as an rx-addr address expression or using register notation (R2 through R12).

The reply vector can be generated using the \$REPLYV macro. If specified, this macro does not return until a valid reply to the WTOR is entered by the operator. If a non-valid reply is issued, a \$HASP299 message is issued and the WTOR is re-issued.

The value of the A-type address constant associated with the reply is returned in R1 if the return code in R15 is zero.

This keyword is only valid if TYPE=WTOR is specified.

It is valid for execute, inline, and modify macro forms. The last value that was set for this parameter will be used for the execute macro form with the NOCHECK parameter specified or the modify macro form with the NOINIT parameter specified (or as the default).

**SEPAR=**

Specifies the separator character that is to be used between fields when the message is displayed. Specify one character enclosed within single quotation marks or the word NULL. NULL indicates no separator. (See the SEPAR= keyword on the \$SCAN macro for further information.)

‘,’

Indicates a comma is used as the message segment separator character.

‘c’

Indicates that the character specified will be used as the separator character. You must code the single quotation marks.

**NULL**

Indicates that no separator character is used.

This keyword is valid for all macro forms. The default is SEPAR=‘,’ for list, inline or execute (with the COMPLETE parameter specified or as the default) macro forms. The last value that was set for this parameter will be used for the execute macro form with the NOCHECK parameter specified or the modify macro form with the NOINIT parameter specified (or as the default).

**DISPRTN=**

Specifies a value for an address of the routine to which \$SCAN gives control to display each line of the message. If this keyword is not specified, the \$MSGDISR routine in module HASPMSG receives control. This keyword may be specified as one of the following values:

- The name of the routine (as on a \$CALL).



- Register notation (R2 through R12). This specifies a register containing the routine address.
- A literal value to load as the routine address.

A second positional parameter allows the definition of the contents of register 11 (R11) on entry to the display routine.

**HCT**

HCT address

**HCCT**

HCCT address

**CR11**

R11 at time of call (or value specified on CR11= parameter)

If not specified, the default values of the second positional are as follows:

**HCT**

If distrtn is found in the PADDR or UPADDR

**HCCT**

If distrtn is found in the CADDR or UCADDR

**CR11**

All other specifications

If you supply your own display routine, it is up to your display routine to process the \$BLDMSG operands set in the \$BLDMSG parameter list and to determine the route and descriptor codes from the high-level \$SCANTAB.

This keyword is valid for execute, inline, and modify macro forms. The last value that was set for this parameter will be used for the execute macro form with the NOCHECK parameter specified or the modify macro form with the NOINIT parameter specified (or as the default).

**CR11=**

Specifies the value that to be placed in register 11 (R11) when invoking exits out of \$BLDMSG.

**DISPER=**

Specifies a display identifier. This identifier is supplied to the \$SCAN macro to determine if a message segment will be constructed. (See the DISPER= keyword on the \$SCAN macro for further information.) This parameter may be specified in one of the following ways:

- A label for a field containing the display ID byte.
- A literal value that specifies the display ID byte.
- Register notation (R2 through R12). The register contains the display ID in the low-order byte.

It is valid for execute, inline, and modify macro forms. The last value that was set for this parameter will be used for the execute macro form with the NOCHECK parameter specified or the modify macro form with the NOINIT parameter specified (or as the default).

**CBADDR=**

Specifies a control block address to be used by \$SCAN. This parameter may be specified as an rx-addr address expression or using register notation (R2 through R12).

It is valid for execute, inline, and modify macro forms. The default is to omit the control block address for the inline or execute (with the COMPLETE parameter specified or as the default) macro forms. The last value that was set for this parameter will be used for the execute macro form with the NOCHECK parameter specified or the modify macro form with the NOINIT parameter specified (or as the default).

**MF=**

Specifies the macro form. Specify one of the following forms:

**I**

Indicates the inline form of the macro.

## **\$BLDMSG**

Use the inline macro form (MF=I) to obtain storage, build an inline parameter list and invoke a service to issue the message.

### **L**

Indicates the list form of the macro.

Use the list macro form (MF=L) together with the execute form (MF=E) of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to set the parameters.

### **E**

Indicates the execute form of the macro.

Use the execute macro form (MF=E) together with the list form of the macro for applications that require reentrant code. The execute form sets the parameters, stores them into the area defined by the list macro form, and invokes a service to issue the message.

#### **rx-addr or (Rn)**

Specifies the address of the storage defined by the list (L) macro form. The list form address may be specified as an rx-addr expression or register notation (R1 through R12).

Optionally, you can specify additional keywords. The keyword values and their meanings are:

#### **COMPLETE | NOCHECK**

Specifies whether \$BLDMSG should check for required parameters and supply defaults for omitted optional parameters (COMPLETE), or whether \$BLDMSG should not check for required parameters and should not supply the omitted optional parameters (NOCHECK). The default is COMPLETE.

### **M**

Indicates the modify form of the macro.

Use the modify form (MF=M) of the macro to modify an already defined \$BLDMSG parameter list.

#### **rx-addr or (Rn)**

Specifies the address of a list (L) macro form parameter list. The list form address may be specified as an address expression or register notation (R1 through R12).

Optionally, you can specify additional keywords. The keyword values and their meanings are:

#### **INIT | NOINIT**

Specifies whether to initialize (INIT) or not (NOINIT) the \$BLDMSG parameter list to the list macro form defaults before setting parameter values. The default is NOINIT.

### **LONG=**

Specifies whether (YES) or not (NO) JES2 will issue the long form of the message. The long form of a message is defined by specifying 'DISPALL=LONGONLY' on the \$SCANTAB macro for certain parts of the message. The text or value represented by \$SCANTAB is displayed in addition to the text for the short form of the message.

### **BRANCH=YES|NO**

Specifies whether (YES) or not (NO) a branch entry WTO is to be done when an MVS WTO is issued. The default is BRANCH=NO.

### **MULTI=YES|NO**

Whether (YES) or not (NO) the display routine is to issue the message as a multi-line WTO. If MULTI=YES is specified, JES2 will display up to 70 characters on a single line; if MULTI=NO is specified, JES2 will display up to 125 characters. For MULTI=NO, if more than one line is to be issued, JES2 will issue the second and subsequent lines as separate messages. MULTI=NO is best used when the expected length of the message is more than 70 but less than 125 characters. The default is MULTI=YES.

## **Register contents when \$BLDMSG is invoked**

### **Register Contents**

**0-10**

Not applicable

**11**

- HCT in JES2 or SUBTASK environment
- HCCT in USER
- HFCT in FSS

**12**

Not applicable

**13**

- PCE in JES2 environment
- DTE in SUBTASK
- Save area in USER or FSS

**14-15**

Not applicable

**Register contents on exit from \$BLDMSG****Register  
Contents****0**

Unpredictable

**1**

The value can be one of the following:

- DOMID if WTO or WTOR
- the address of the REPLY processing data if REPLYV= was specified
- Unpredictable

**2-13**

Unchanged

**14**

Unpredictable

**15**

Return code.

**Return codes**

The following return codes (decimal) are returned in register 15.

**Return Code  
Meaning****0**

Processing successful (no errors)

**4**

Indicates that SCAN found an obsolete keyword (as indicated by a \$SCANTAB entry specifying OBS=YES).

**8**

Indicates that SCAN found a keyword not supported in the tables.

**12**

Indicates that SCAN encountered scanning errors (for example, non-valid syntax) that could not be resolved.

## Usage notes

1. MSGID= must be specified on a list or modify macro form before using the execute macro form with the NOCHECK parameter specified.
2. TYPE= must be specified on a list or modify macro form before using the execute macro form with the NOCHECK parameter specified.
3. REPLY=, REPLEN=, and ECB= are required with TYPE=WTOR, and these parameters must be coded on the same macro form.
4. If you code \$BLDMSG in an exit, you must also specify the parameter list DSECT (\$BLDMSGGL) on the \$MODULE macro instruction.
5. If you have coded a USING statement for symbol BLD, issue a corresponding DROP statement before coding the \$BLDMSG macro.

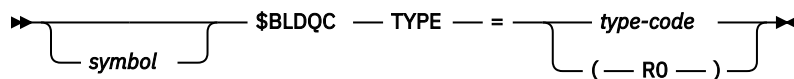
## Environment

- JES2 main task or during JES2 initialization or termination.
- \$WAIT can occur.
- Can be invoked in any environment (JES2, SUBTASK, USER, FSS)

## \$BLDQC – Call the quick cell build/extend routine

Use \$BLDQC to call the quick cell build/extend routine to build or extend a quick cell pool.

## Format description



### TYPE=

Specifies the type of quick cells to build.

### type-code

specifies the type code as defined in the \$QCTGEN macro for the quick-cell type to build. Quick cell types are defined as one of the following:

#### Type-Code

##### Meaning

#### SAVE

Standard save area for MVS linkage conventions as described in [z/OS MVS Programming: Assembler Services Guide](#)

#### JIB

JOE information block

#### BUF

Standard 4K buffer

#### RPL

Request parameter lists for GETDS processing

#### GETRC

Control block areas for GETRC processing

### (RO)

Specifies the register that contains the type-code; if coded, be certain that the two low-order bytes of the register contain the quickcell type-code as defined in the \$QCTGEN macro; the two high-order bytes must be zeroed.

The TYPE= keyword must be specified.

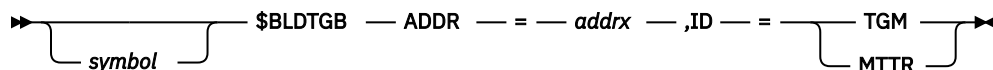
## Environment

- Functional subsystem (HASPFSM).
- MVS WAIT can occur.

## \$BLDTGB – Queue TGBs to the HASPOOL processor

Use \$BLDTGB to build track group blocks (TGBs) and queue them off the \$SPOOLQ in the HCT. The TGB represents a bad track group for which the HASPSPOL processor attempts recovery.

### Format description



#### ADDR=

Specifies the address of the track group map (TGM) that contains bad track groups or the MTTR (JES2 spool track address) of a single bad track group.

#### ID=

Specifies whether the ADDR= keyword specifies a TGM or MTTR.

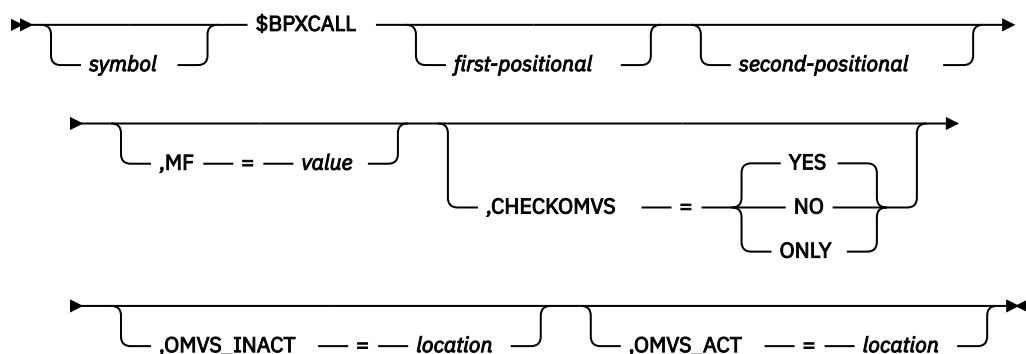
## Environment

- Main task.
- \$WAIT cannot occur.

## \$BPXCALL – Call omvs services

Generates a proper calling sequence to get into BPX services for a given call type. The main purpose is to generate the right stub code to get the address of the service routine being called (so that JES2 does not have to link-edit or LOAD the stub routines).

### Format description



#### 1st positional

Indicates the service to be called. Supported are:

##### BPX1CLO

close()

##### BPX1HST

gethostname(), gethostid()

##### BPX1MPC

mvsprocclp()

## **\$CALL**

### **BPX1SDD**

set\_dub\_default()

### **BPX1SOC**

socket(), socketpair()

### **2nd positional**

Indicates the remaining parameters. These values are passed directly to the CALL macro for the service specified. See the z/OS UNIX System Services documentation for the proper parameters for each service.

### **MF=**

The MF= value for the \$CALL

### **CHECKOMVS=**

Indicates whether the calling sequence is to check for z/OS UNIX System Services activity before the call. Valid values are:

#### **YES**

Indicates that the code is to check for z/OS UNIX System Services active before calling the BPX service. YES is the default.

#### **NO**

Indicates that the code is not to check for z/OS UNIX System Services active before calling the BPX service.

#### **ONLY**

Indicates that only the sequence of code to check for z/OS UNIX System Services active is to be generated.

### **OMVS\_INACT**

Specifies the location to receive control if z/OS UNIX System Services is not active.

### **OMVS\_ACT**

Specifies the location to receive control if z/OS UNIX System Services is active. PARM0 is only valid when CHECKOMVS=ONLY.

## **Environment**

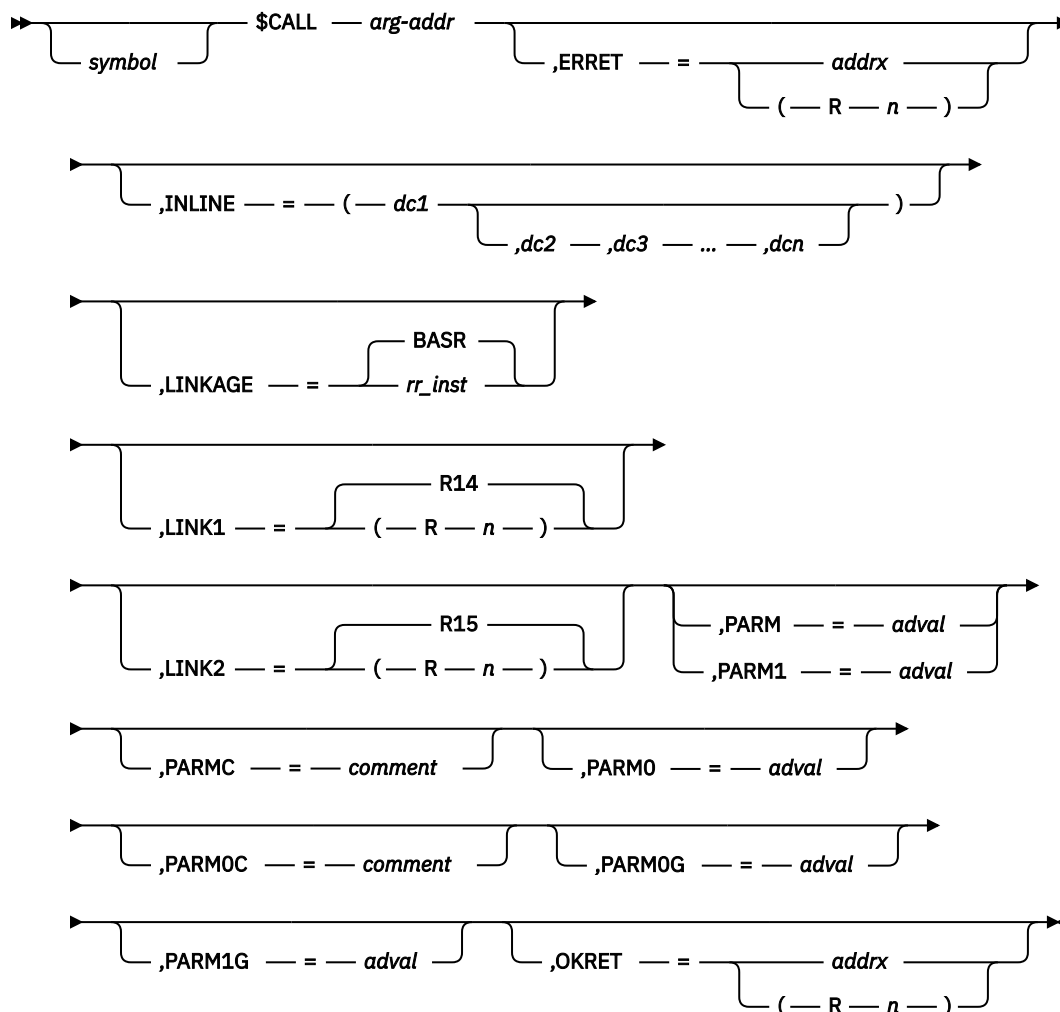
JES2 main task only.

## **\$CALL – Call a subroutine from JES2**

---

Use \$CALL to call a subroutine from a JES2 module. Note that \$CALL will attempt to branch to a global routine before attempting to branch to a local one. Therefore, ensure that, if you have defined two routines with the same name, your routine branches to the desired one.

## Format description



### arg-addr

Use 'arg-addr' to specify one of the following arguments (JES2 will search for and process these arguments in the order listed):

1. A register containing an address to which this macro will branch. (If register notation is used, the designated register must contain the address of the subroutine to be called before executing \$CALL.)
2. The name (label) of a routine listed in one of the following tables:

#### Table Address Prefix

##### CADDR

C@

##### PADDR

P@

##### UCADDR

UC@

##### UPADDR

UP@

JES2 will search through the tables in the above order.

#### Note:

## \$CALL

- a. The UPADDR is chained out of \$UPADDR in the \$HCT.
  - b. The UCADDR is chained out of the CCTUCADD in the \$HCCT.
3. A label pointing to either:
- A name (label) of a local routine this macro will call; this causes \$CALL to generate an ADCON.
  - A label of a field containing the address of a routine this macro will call
4. The name of a routine contained in another module; this causes \$CALL to generate a VCON.

### LINKAGE=

Specifies the assembler instruction (RR-type) to use for this \$CALL. The default is BASR.

### LINK1=

Specifies the register to be used as the first operand (that is, R1) on the BASR (branch and save registers) assembler instruction. This register provides linkage to the called routine. R14 is the default or R0 is the default if LINKAGE=BAKR. If LINK1=NONE is specified, it indicates that, for LINKAGE=PC or BAKR, the address of the inline parameter list should not be loaded into a register. The called routine can use an ESTA to obtain the return address from the linkage stack.

### LINK2=

Specifies the register to be used as the second operand (that is, R2) on the BASR (branch and save registers) assembler instruction. This register provides linkage to the called routine. R15 is the default.

### PARM= | PARM1=

Specifies a parameter value that is to be passed to the called subroutine through register 1. You can only use a register, label, or assembler literal address. If you use register notation, the designated register must contain the address of the parameter value.

### PARM0=

Specifies a parameter value that is to be passed to the called subroutine through register 0. You can only use a register, label, or assembler literal address. If you use register notation, the designated register must contain the address of the parameter value.

### PARMC=

Specifies the assembler comment that JES2 passes to the decoding routine when processing the PARM= or PARM1= value.

### PARMOC=

Specifies the assembler comment that JES2 passes to the decoding routine when processing the PARM0= value.

### PARMOG=

Specifies a parameter value that is to be passed to the called subroutine through register 0. You can use the entire 64-bit register.

### PARM1G=

Specifies a parameter value that is to be passed to the called subroutine through register 1. You can use the entire 64-bit register.

### INLINE=

Specifies the operands that comprise an inline parameter list. For example:

```
, INLINE=(AL4(USER), B '&FLAG', CL4 ' PARM')
```

specifies the following inline parameter list:

```
DC AL4(USER)
DC B '&FLAG'
DC CL4 ' PARM'
```

**Note:** &FLAG will be substituted if the variable is defined.



**ERRET=**

Specifies the register (2-12) or address of the error routine that is to receive control if a nonzero return code is returned in R15.

**OKRET=**

Specifies the address of the routine that is to receive control when JES2 passes back a zero return code in R15.

## Programming considerations

\$CALL linkage is determined by the LINKAGE= parameter. If you use register notation to specify the arg-addr parameter (that is, \$CALL (Rx) ) and you invoke \$CALL in AMODE 24, be certain the high-order byte of this register does not contains data.

Some JES2 services are running in 64-bit addressing mode. These services, regardless of whether they are called directly or invoked by a macro, need register 11 to contain a 64-bit pointer to the HCT, HCCT, or HFCT. When JES2 invokes an exit, it ensures that register 11 is a valid 64-bit pointer. Because exits should not need to know which services are running in 64-bit addressing mode, the invoked exit should not corrupt the high order 33 bits of register 11 before invoking any JES2 service.

## Environment

- Main task, subtask, FSS, and user address space.
- \$WAIT can occur depending on the routine that is called.
- Callers in AR ASC mode are supported. For PARM0= and PARM1=, the appropriate access registers are set if ASC mode is not primary.

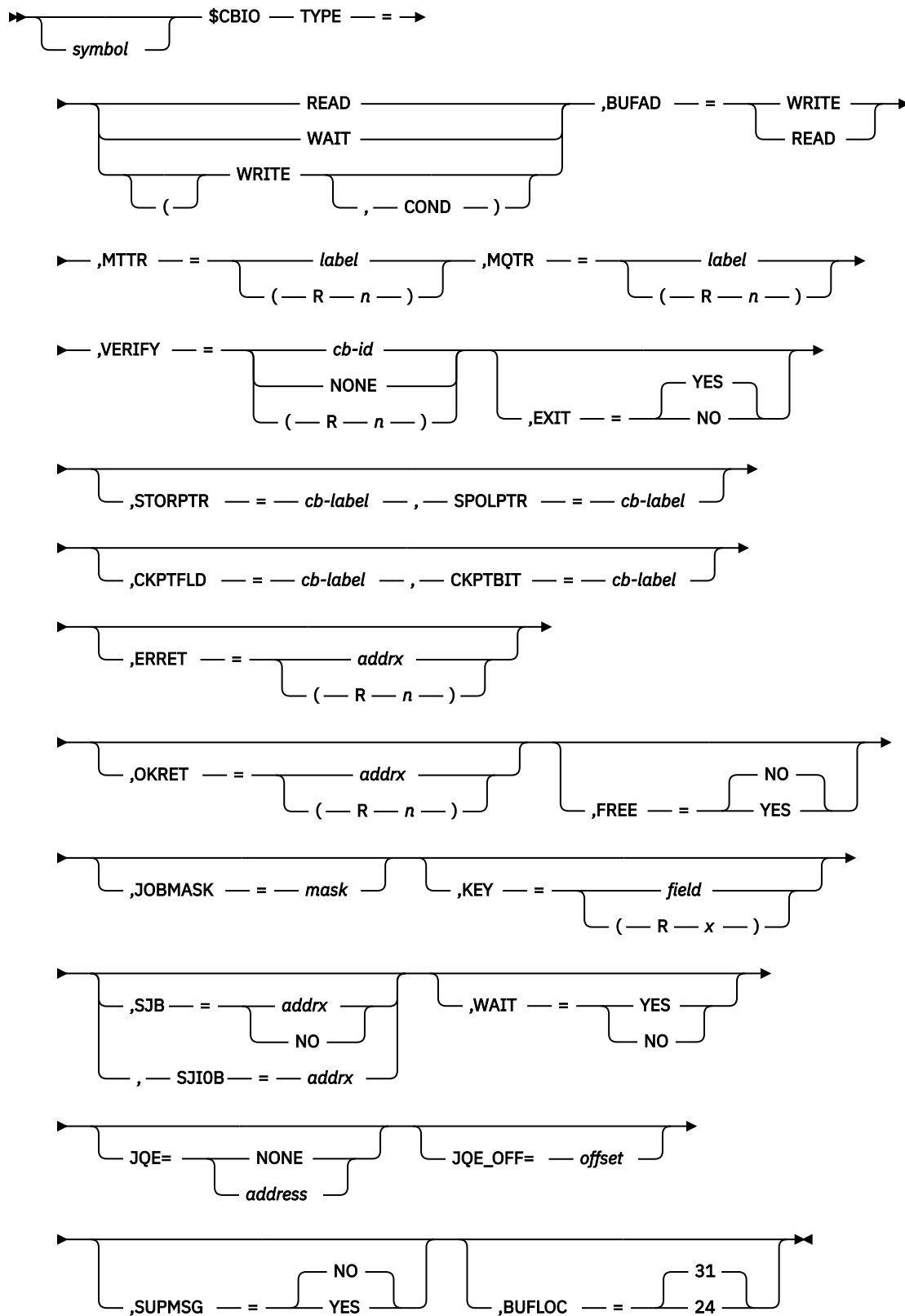
## \$CBIO – Control block I/O routine

---

Use \$CBIO to access the control block I/O (\$CBIO) service routine to perform I/O for JES2 control blocks.

**Note:** The caller must hold the SJB lock and cannot hold *any* other MVS lock. The address of the SJB must be in register 10 prior to calling \$CBIO, unless you specify the SJB= or SJI0B= keywords on the \$CBIO macro.

## Format description



### TYPE=

Specifies the type of I/O operation that is to be performed. This is a required parameter.

**READ**

Indicates this is an I/O read.

**WAIT**

Waits for the completion of a \$CBIO request that specified WAIT=NO.

In a JES2 environment, when a \$CBIO TYPE=READ or TYPE=WRITE request is made with WAIT=NO, verification processing and buffer management occurs when the I/O completes. No additional code is required of the caller of \$CBIO. However, in a non-JES2 environment, this processing cannot be done automatically. In this case, a \$CBIO TYPE=WAIT is needed to wait for the I/O to complete and then perform the required verification and buffer management. Therefore, TYPE=WAIT is only valid in a non-JES2 environment.

If WAIT=NO is specified in a non-JES2 environment, a \$CBIO TYPE=WAIT call may be needed to complete verification of the control block and to perform any needed buffer management (including any error processing).

**Note:** For TYPE=WAIT, the parameters BUFAD=, MTTR=, VERIFY=, KEY=, STORPTR=, SJB=, WAIT=, FREE=, SPOLPTR=, JOBMASK=, CKPTFLD=, and CKPTBIT= are not valid. The values specified on the TYPE=WRITE or TYPE=READ are used. SJIOB= is required if TYPE=WAIT.

**WRITE**

Indicates this is an unconditional I/O write.

**(WRITE,COND)**

Indicates this is a conditional I/O write. That is, JES2 writes a control block only if the checkpoint bit (BUFM1CKP bit in byte BUFMFLG1) is on.

Specifying COND is mutually exclusive with the CKPTFLD and CKPTBIT parameter pair.

**BUFAD=****WRITE THE BUFFER (TYPE=WRITE)**

Specifies the label of, or a register that contains, the address of the input/output block (IOB) that corresponds to the buffer that is to be written to SPOOL. For TYPE=WRITE, you must also specify BUFAD=.

**READ THE BUFFER (TYPE=READ)**

Specifies the label of, or a register that contains, the address of the input/output block (IOB) that corresponds to the buffer that is to be read from SPOOL. For TYPE=READ, specifying BUFAD= is "optional" on the \$CBIO macro. If you do NOT specify BUFAD= or specify BUFAD=0, then the \$CBIO routing gets the buffer for you and puts its address into register 1 when it returns control. BUFAD is not allowed with TYPE=WAIT.

**Note:** Whether writing or reading the buffer:

1. If you specify a register for BUFAD=, the address of the IOB must be loaded into this register before executing the \$CBIO macro.
2. When your I/O operations are complete, you are responsible for "freeing" the buffer you have used for your \$CBIO operations. This is normally accomplished with the \$FREEBUF macro. For more information on the \$FREEBUF macro see ["\\$FREEBUF – Return a JES2 buffer to the JES2 buffer pool" on page 172.](#)

**MTTR=**

Specifies the label of, or a register that contains, the spool track address of the record to be read or written. If you specify a register, the spool MTTR (module track record) must be loaded into the designated register before executing this macro instruction. MTTR is required for TYPE=READ or TYPE=WRITE, but is not allowed with TYPE=WAIT.

**MQTR=**

Specifies the 6 byte MQTR SPOOL address of the record to read or write. If a register is used, it must contain the MQTR (0000mmtt tttttrr). MQTR is required for TYPE= READ or TYPE=WRITE, but is not allowed with TYPE=WAIT. MQTR is not allowed in the JES2 environment.

**Note:** MQTR is mutually exclusive with MTTR.

**VERIFY=**

Specifies the control block identifier (cb-id) or a register that contains the address of the cb-id that is used to validate the control block passed to Exit 7 or 8. Specify the cb-id as a 4-byte EBCDIC value. If you specify NONE, JES2 will not verify the control block after the read; however, job key validation should be done. VERIFY is required if TYPE=READ or TYPE=WRITE, but is not allowed with TYPE=WAIT.

**EXIT=**

Specifies whether (YES) or not (NO) this macro instruction will call \$EXIT7 or \$EXIT8. EXIT=YES is the default.

**Note:** Because you can use this macro instruction in an installation-defined exit, and \$CBIO calls Exit 8, you can use this keyword to allow or disallow recursive exit calls, as required.

**STORPTR=**

Specifies the offset of the chaining field that is used to perform a series of control block I/Os. For example, use this keyword if you need to write an entire IOT chain to spool. If you do not specify STORPTR=, only a single buffer (as specified by MTTR=) is either read or written. If STORPTR= is specified, SPOLPTR= is also required. The default for STORPTR is a null value (unspecified). STORPTR is not allowed with TYPE=WAIT. Blocks are written in the reverse order of the chain, that is, they are written from the newest to the oldest. If A points to B points to C, then the blocks are written in the order CBA.

**SPOLPTR=**

Specifies the offset of the field that is used to obtain the MTTR for a series of control block I/Os. For control block READ it is the offset of the field that contains the MTTR of the next control block. For control block WRITE it is the offset of the field that contains the MTTR of the current control block. For example, use this keyword if you need to write an entire IOT chain to spool. If SPOLPTR is not specified, only the single control block identified by MTTR= will be used. The default for SPOLPTR is a null value (unspecified). If STORPTR= is specified, SPOLPTR= is also required. SPOLPTR is not allowed with TYPE=WAIT.

**CKPTFLD=**

Specifies the offset into the buffer that is used to determine if this buffer is to be checkpointed. The bit (as specified by CKPTBIT=) must be set to one before writing the buffer to spool, if checkpointing is required. If the bit is not set on, and a chain of writes is not requested, \$CBIO will not write this buffer to spool. If the bit is not set on and a chain of writes is requested, \$CBIO will not write this buffer to spool and will check the next buffer (as pointed to by STORPTR=). If TYPE=WRITE and CKPTFLD was not specified, the buffer, or chain of buffers, will be unconditionally written. However, if CKPTFLD was specified, whether or not the buffer is written depends on the setting of CKPTBIT. If CKPTFLD= is specified, CKPTBIT= is required.

Specifying CKPTFLD is mutually exclusive with TYPE=(WRITE,COND).

**CKPTBIT=**

Specifies the bit to be checked in the field specified by CKPTFLD=. If this bit is set to one, the buffer is written to spool; if this bit is set to zero, the buffer is not written to spool. If CKPTFLD= is specified, CKPTBIT= is required.

Specifying CKPTBIT is mutually exclusive with TYPE=(WRITE,COND).

**ERRET=**

Specifies the label of, or a register that contains, the address of a routine that receives control if the I/O operation was unsuccessful (that is, if register 15 contains a nonzero return code). A chain of control blocks will not be written if an error is encountered during the I/O for that chain. ERRET= is optional, and either a label or register notation may be used to specify the error routine address.

**OKRET=**

Specifies the address of the routine that is to receive control when the I/O operation is successful (register 15 contains 0). You can specify a label that corresponds to the routine's address or a register that contains the address of the routine.

**FREE=**

Specifies whether or not the buffer should be freed after it is written, or after an error occurred. The default is NO. FREE= is mutually exclusive with TYPE=WAIT.

**JOBMASK=**

Specifies the job mask that will be used when calling exit 7. The job mask determines whether the exit can be taken for a particular job. If no job mask was specified, the exit will be taken without one. If the JCT is being read in, the job mask will be taken from this JCT. FREE= is mutually exclusive with TYPE=WAIT.

**JQE=**

Specifies one of the following values:

- Label or register that contains the address of the JQE
- NONE - which indicates that there is no related JQE for this I/O operation.

If the JQE address is not provided explicitly, but you can determine the JQE address because one of the following parameters is specified:

- KEY=JQEJBKEY
- MTTR=JQETRAK
- KEY=IOTJBKEY
- MTTR=IOTTRACK
- KEY=JCTJBKEY
- MTTR=JCTTRAK

JES2 will use the implicitly specified JQE/JQE\_OFF.

If you do not specify JQE= nor JQE\_OFF=, JES2 uses the JQE address contained in the PCEJQE. JQE= is mutually exclusive with JQE\_OFF=.

JQE= is only valid in the JES2 environment.

**JQE\_OFF=**

Specifies a label or register that contains the offset of the JQE.

If the JQE address is not provided explicitly, but you can determine the JQE address because one of the following parameters is specified:

- KEY=JQEJBKEY
- MTTR=JQETRAK
- KEY=IOTJBKEY
- MTTR=IOTTRACK
- KEY=JCTJBKEY
- MTTR=JCTTRAK

JES2 will use the implicitly specified JQE/JQE\_OFF.

If you do not specify JQE= nor JQE\_OFF=, JES2 uses the JQE address contained in the PCEJQE. JQE\_OFF= is mutually exclusive with JQE=.

JQE\_OFF= is only valid in the JES2 environment.

**KEY=**

Specifies the key field of the control block that will be verified by \$VERIFY. If a register is used, it must contain the address of the field. If a key is not specified, only the control block identifier will be used to verify the control block. KEY= is mutually exclusive with TYPE=WAIT.

**SJB=**

Specifies either the address of the SJB, or NO. NO indicates that no SJB was specified, and the default (SJB address in register 10) is not to be used. If neither an SJB nor an SJIOB is specified, an SJIOB

## **\$CBIO**

will be obtained and initialized. This parameter is used in the user environment only. SJB= is mutually exclusive with TYPE=WAIT.

### **SJIOB=**

Specifies the address of the SJIOB. If specified, the SJIOB address will be loaded into register 10 and the SJB parameter will be ignored. This parameter is used in the user and subtask environments only.

### **WAIT=**

Specifies whether or not the \$CBIO service routine is to cause the caller to wait until the I/O operation completes (WAIT=YES). If not (WAIT=NO), control returns to the caller.

### **SUPMSG=**

In JES2 environment specifies whether DISTERRs and dump are suppressed for an error condition discovered by CBIO.

In non-JES2 environment specifies whether messages HASP363, HASP364, HASP370 and the symptom record should be suppressed if there is an error condition discovered by CBIO.

NO is the default. YES means suppress.

### **BUFLOC=**

Specifies the addressability that a new buffer should have.

#### **24**

Indicates if the buffer must be in 24-bit memory.

#### **31**

Indicates if the buffer can be in 24-bit or 31-bit memory.

## **Register contents when \$CBIO is invoked**

### **Register**

#### **Contents**

#### **2-9**

Not applicable

#### **11**

HCT/HCCT/HFCT base address - as appropriate

#### **12**

Not applicable

#### **13**

PCE/Save area address - as appropriate

#### **14-15**

Not applicable

## **Register contents on exit from \$CBIO**

### **Register**

#### **Contents**

#### **0**

Unchanged

#### **1**

READ - Address of buffer. If there was more than one READ operation, the register contains the address of the first buffer read.

WRITE - unchanged

#### **2-9**

Not applicable

#### **10**

User environment - SJB or SJIOB

JES2 environment - not applicable

- 14** Return address
- 15** Return code

## Return codes

The following return codes (in decimal) are returned in register 15.

### Return Code Meaning

- 0** Processing completed successfully.
- 4** Processing failed because the control block was not valid.
- 8** Processing failed due to an I/O error.
- 12** Processing failed because the track address was not valid.
- 16** Processing failed because:
  - READ - JES2 was unable to obtain a buffer
  - WRITE - no buffer was passed to the \$CBIO service
- 20** Processing failed because the caller did not hold the subsystem SJB lock. (Applicable in the user environment only.)
- 24** Processing failed because the control block read is of an unidentifiable type. JES2 could not verify the control blocks.
- 28** Processing failed because JES2 could not obtain storage for the SJIOB. (Applicable in the user environment only.)
- 32** Processing failed because the buffer does not have a valid buffer address table (BAT) entry. (Applicable in the JES2 maintask or subtask environment only.)

## Environment

- JES2 main task, JES2 subtask, and user environment.
- \$WAIT can occur.
- MVS WAIT will occur in user environment.

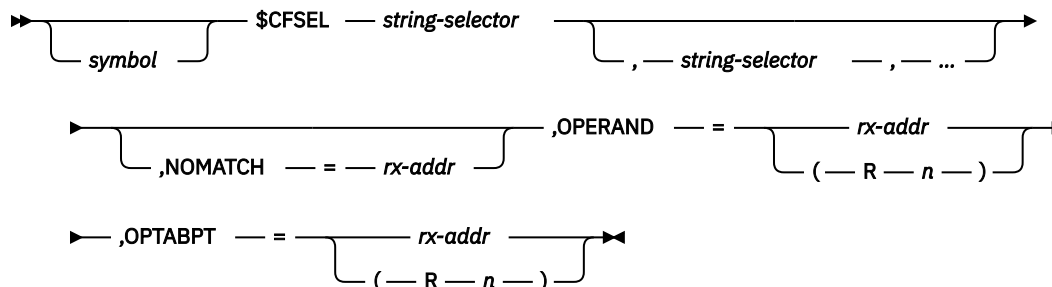
## **\$CFSEL – Select label to process a command operand string**

---

\$CFSEL is intended for use with the JES2 Exit 5 command preprocessor.

\$CFSEL compares a command operand string to a set of one or more specifications, called string-selectors, that you provide. If one of the string-selectors matches the command operand, \$CFSEL branches to the label specified on that string-selector. If none of the string-selectors match the command operand string, \$CFSEL branches to another label, or if no label is provided, processing continues at the instruction following \$CFSEL.

## Format description



### string-selector

The string-selector is a required positional parameter. You must code one or more string-selectors. Each string-selector provides specifications for the command operand string and a label to which \$CFSEL is to branch if the command operand string matches the specifications. The command operand string is explained under the description of the **OPERAND=** parameter. There are three forms of the string-selector:

**('string',label)**

**('string',label,min-residual,max-residual)**

**('string',label,EXACT)**

A string-selector must define a character string, 'string', and a label. Optionally, a string-selector can also define a length specification. You define a length specification by coding *min-residual* and *max-residual*, or by coding **EXACT**.

Starting with the first string-selector, \$CFSEL compares each string-selector to the command operand string. The first string-selector that matches causes \$CFSEL to branch to the label specified on that string-selector. \$CFSEL ignores any remaining string-selectors. For a string-selector to match a command operand string:

- 'string' must match a like number of characters in the command operand string beginning with the character pointed to by the **OPERAND=** parameter.
- If the command operand string contains residual characters, they must satisfy any length specification coded on the string-selector. Residual characters are those characters in the command operand that follow the characters that are compared to 'string'. For example, if the command operand contains **Q=ABCD** and 'string' is specified as '**Q=**', the characters **ABCD** are residual characters.

If none of the string-selectors match the command operand string, \$CFSEL branches to the address specified on the **NOMATCH=** parameter. If that parameter is omitted, control returns to the instruction following \$CFSEL.

### 'string'

The character string that \$CFSEL compares to the command operand string pointed to by the **OPERAND=** parameter. The string must be enclosed in single quotation marks and must follow the rules for character constants. The length of the string must not exceed 255 characters.

This parameter is required and has no default.

### label

The label of the instruction where \$CFSEL is to branch if the string-selector matches the command operand string. The label name must be valid for use as an A-type address constant.

This parameter is required and has no default.

### min-residual

Specifies the minimum number of residual characters that the command operand string must contain. *min-residual* can be any value from 0 through 255.

The default value is 0.



*min-residual* is mutually exclusive with **EXACT**.

#### **max-residual**

Specifies the maximum number of residual characters that the command operand string may contain. *max-residual* can be any value from 0 through 255.

The default value is 255.

*max-residual* is mutually exclusive with **EXACT**.

#### **EXACT**

Specifies that 'string' must exactly match the characters in the command operand string and that the command operand string must contain no residual characters. Coding EXACT is the equivalent of coding 0 for both *min-residual* and *max-residual*.

**EXACT** is mutually exclusive with *min-residual* and *max-residual*.

There is no default value. If you omit **EXACT**, the *min-residual* and *max-residual* default values apply.

#### **NOMATCH=**

Specifies the address, in the form of an RX-type address, that is to receive control if none of the string-selectors match the command operand string. If you omit this parameter and none of the string selectors match the command operand string, control returns to the instruction following \$CFSEL.

#### **OPERAND=**

Points to the command operand string that \$CFSEL is to compare to each string-selector. The **OPERAND=** parameter can specify an RX form of address or can use register notation (R1-R12).

- If you specify an RX form of address, **OPERAND=** points to a full-word that points to the command operand string,
- If you use register notation, the specified register points to the command operand string. Register notation requires that the register contain the address of the command operand string at the time you issue \$CFSEL.

This parameter is required and has no default.

The register specified by **OPERAND=**, or the full-word that is pointed to by **OPERAND=** points to the first character of the command operand string. The command operand string includes that character and all following characters to either the next comma that is not enclosed in a quoted string or, if there is no comma, to the end of the command.

The following examples show how \$CFSEL interprets the command operand string.

1. If the command contains **\$O Q,Q=A,CANCEL and**
  - If **OPERAND=** points to the character **Q**, the command operand string consists of the characters **Q=A**.
  - If **OPERAND=** points to the first character **C** in **CANCEL**, the command operand string consists of the characters **CANCEL**.
2. If the command contains **\$DMR5,'HELLO, EVERYONE'**
  - If **OPERAND=** points to the first R, the command operand consists of the characters **R5**.
  - If **OPERAND=** points to the first character **'**, the command operand string consists of the characters **'HELLO, EVERYONE'**.

**Attention:** The command operand string must be at the address it occupied when it was parsed by JES2. If you move the command operand string before you invoke \$CFSEL, \$CFSEL will not work correctly.

#### **OPTABPT=**

Points to the current entry in the command operand pointer table that JES2 provides to Exit 5. You can use an RX form of address or use register notation (R1-R12).

## \$CFSEL

- If you specify an RX form of address, **OPTABPT=** points to a full-word that points to the current entry in the command operand pointer table.
- If you use register notation, the specified register points to the current entry in the command operand pointer table. Register notation requires that the register contain the address of the command operand pointer table at the time you issue \$CFSEL.

This parameter is required and has no default.

## Register contents when \$CFSEL is invoked

### Register Contents

#### 0-10

Not applicable

#### 11

HCT base address

#### 12

Not applicable

#### 13

PCE base address

#### 14-15

Not applicable

## Register contents on exit from \$CFSEL

### Register Contents

#### 0

Unpredictable

#### 1

Points to the command operand string identified by the **OPERAND=** parameter.

#### 2-13

Unchanged

#### 14

If there is a match with a string selector, the address where \$CFSEL will branch, otherwise zero.

#### 15

If the command operand string matched one of the string-selectors, R15 contains the length of the matching string. Otherwise, R15 contains zero.

## Return codes

None. Register 15 contains the results.

## Usage notes

1. The command operand string (specified with OPERAND=) will not match if its length (computed from the OPTABPT= parameter) exceeds 255 characters.
2. The length of the matching string, or zero if no match, is set in R15 on exit from this macro. This length value is also returned in the COMSSLEN field in the command processor PCE.
3. JES2 returns the residual character length in the COMSRLEN field of the command processor PCE:
  - If one of the string-selectors results in a successful match, the residual character length is the length of the command operand string minus the length of the 'string' that compared successfully to the command operand string.

- If none of the string-selectors result in a successful match, the residual character length is the length of the command operand string.
4. This macro **must** execute under a command processor PCE. This macro also assumes the command operand input string is part of a JES2 command that the JES2 command processor parsed.

## Environment

- JES2 main task under a command processor PCE only.
- \$WAIT cannot occur.

## Examples

The following examples show various forms of string-selectors and the results when they are matched with specific command operand strings. The last example contains a code segment that shows Exit 5 logic and structure when \$CFSEL is used. The last example also shows how to code multiple string-selectors.

### Example 1

This string-selector matches on Q=*anything*. The command operand string can contain from 0 to 255 residual characters. If this string-selector is the first to match, \$CFSEL branches to label CLASSRTN.

```
$CFSEL ('Q=',CLASSRTN),...
```

**Command Operand String**  
**Result of Comparison**

**Q=**  
Matches

**Q=A**  
Matches

**Q=AB**  
Matches

**Q=ABCDEFGHIJ**  
Matches

### Example 2

This string-selector matches on Q=x where x is any single character. The command operand string must contain one residual character and no more. If this string-selector is the first to match, \$CFSEL branches to label CLASSRTN.

```
$CFSEL ('Q=',CLASSRTN,1,1),...
```

**Command Operand String**  
**Result of Comparison**

**Q=**  
Does not match because the command operand string fails the minimum length specification for residual characters: there are no residual characters.

**Q=A**  
Matches

**Q=AB**  
Does not match because the command operand string fails the maximum length specification for residual characters: there are two residual characters, **AB**

### Example 3

This string-selector matches on Q=x where x is any character string one to sixteen bytes long. The command operand string must contain from 1 to 16 residual characters. If this string-selector is the first to match, \$CFSEL branches to label CLASSRTN.

```
$CFSEL ('Q=',CLASSRTN,1,16),...
```

Command Operand String	Result of Comparison
------------------------	----------------------

<b>Q=</b>	Does not match because the command operand string fails the minimum length specification for residual characters: there are no residual characters.
-----------	---

<b>Q=1</b>	Matches
------------	---------

<b>Q=12</b>	Matches
-------------	---------

<b>Q=123456789ABCDEFGG</b>	Matches
----------------------------	---------

<b>Q=123456789\$ABCDEF#</b>	Does not match because the command operand string fails the maximum length specification for residual characters: there are 17 residual characters, <b>123456789\$ABCDEF#</b>
-----------------------------	---

### Example 4

These string-selectors match when the command operand string contains only the string **CANCEL**. The command operand string can contain no residual characters. If one of these string-selectors is the first to match, \$CFSEL branches to label **CANRTN**.

```
$CFSEL, ('CANCEL',CANRTN,EXACT), ...
      -or-
$CFSEL ('CANCEL',CANRTN,0,0), ...
```

Command Operand String	Result of Comparison
------------------------	----------------------

<b>CANCE</b>	Does not match because the command operand string does not contain the string <b>CANCEL</b> .
--------------	---

<b>CANCEL</b>	Matches
---------------	---------

<b>CANCELQ=M</b>	Does not match because the command operand string contains the residual characters, <b>Q=M</b>
------------------	--

### Example 5

This string-selector matches when the command operand string contains the character string **DEVICE=** and that string is followed by at least one more character but no more characters than the assembler assigned length of the label DCTDEVN. If the string-selector matches, \$CFSEL branches to label **DEVSET**.

```
$CFSEL ('DEVICE=',DEVSET,1,L'DCTDEVN),...
```

### Example 6

This example shows an order dependency because the first string selector is a subset of the second string-selector. The first selector string matches and \$CFSEL branches to DFORMRTN if the command operand string contains the string **FORM=\*\*\*\***. The second selector string matches and \$CFSEL

branches to DEVSET if the number of characters following **FORM=** in the command operand string is a minimum of 1 and a maximum of the assembler assigned length of JOEFORM.

```
$CFSEL ('FORM=****',DFORMRTN,EXACT),
        ('FORM=',DEVSET,1,L'JOEFORM'),...
```

## Example 7

This example shows how an Exit 5 routine might use \$CFSEL. The example assumes the following register values:

```
R5 - Pointer to operand table entry for current operand
R6 - Contains 4 (increment for BXH or BXLE)
R7 - Pointer to operand table entry for last operand

:
NEXTOP  BXH  R5,R6,DONE    Get next operand, if none, branch to done
        L    R2,0(R5)      Address of next operand
$CFSEL  ('DOIT=Y',DORTN,EXACT), Select routine
        ('DOIT=YES',DORTN,EXACT),
        ('DOIT=N',DONTRTN,EXACT),
        ('DOIT=NO',DONTRTN,EXACT),
        ('NAME=',NAMERTN,1,L'XYZNAME),
        NOMATCH=INVORTN,   Handle an operand that does not match
        OPERAND=(R2),      Points to operand
        OPTABPT=(R5)       Operand table entry

DORTN   DS    0H          Process DOIT=Y|YES
        ...
        B    NEXTOP       Loop for next operand

DONTRTN DS    0H          Process DOIT=N|NO
        ...
        B    NEXTOP       Loop for next operand

NAMERTN DS    0H          Process NAME=value
        AL   R2,COMSSLEN   Point to value
        L    R3,COMSLEN    Get length of value
        ...
        B    NEXTOP       Loop for next operand

INVORTN DS    0H          Handle case where operand does not match
        ...

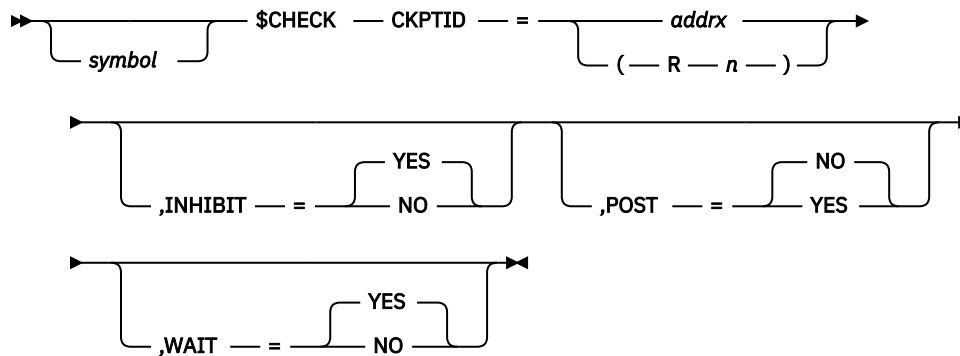
DONE    DS    0H          No more operands
        ...
```

## **\$CHECK – Check checkpoint write completion**

Use \$CHECK to check for the completion of a specific checkpoint write. You can identify the write by a specific checkpoint identifier (token) associated with each write, and based on the status of that write:

- \$WAIT for a resource (CKPTP) indicating I/O is scheduled but not completed
- \$WAIT for a resource (CKPT) indicating that the I/O is not yet scheduled
- Return control immediately to the caller indicating that the I/O is already completed

## Format description



### **CKPTID=**

Specifies the address or register where the address of the 4-byte identifier (token) is stored when returned by the \$CKPT service routine. See the macro “[\\$CKPT – Schedule an element checkpoint](#)” on [page 81](#) for information about obtaining this token. If register notation is used, the address of the checkpoint ID must be loaded into the designated register before executing this macro instruction. This keyword is required.

**Note:** If an invalid token is specified, \$CHECK will issue a \$ESTAE and a \$ERROR, and attempt recovery by waiting for the next scheduled checkpoint write.

### **INHIBIT=**

Specifies whether the \$WAIT issued by this macro allows the processor that issues this macro to be immediately dispatched if it is specifically POSTed (\$POST).

#### **YES**

The \$WAIT is inhibited (not to be \$POSTed).

#### **NO**

The \$WAIT is to be \$POSTed.

### **POST=YES|NO**

Specifies whether the CKPT processor is to be posted or not.

#### **YES**

The CKPT is to be posted.

#### **NO**

The CKPT is not to be posted. POST=NO is the default.

### **WAIT=YES|NO**

Specifies whether (YES) or not (NO) to wait for the checkpoint write. WAIT=YES is the default.

**Note:** If WAIT=NO is specified, INHIBIT= has no meaning and must not be specified.

**Note:** If WAIT=NO is specified, POST= has no meaning and must not be specified.

The following return codes (in decimal) are returned in register 15:

### **Return Code**

#### **Meaning**

**0**

Checkpoint write completed (return to caller)

**4**

Checkpoint write not completed.

**Note:** If INHIBIT=YES is specified, JES2 will not return an RC=4.

## Environment

- Main task.

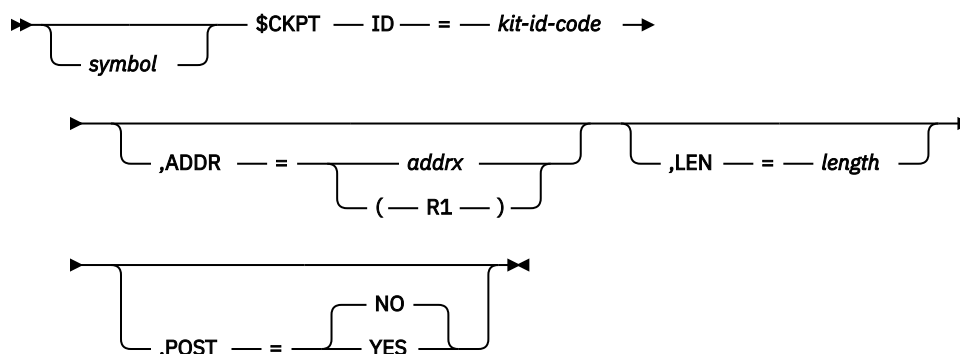
- \$WAIT can occur.

## \$CKPT – Schedule an element checkpoint

Use \$CKPT to schedule the checkpoint of an element in a JES2 checkpoint table that has been altered.

**Note:** \$DOGJQE must be used to checkpoint updates to the \$JQE.

### Format description



#### ID=

Specifies the checkpoint information table (KIT) to be used. This value must be the 1- to 4-character identifier in the KIT for the element to be checkpointed. If more than 4 characters are specified, only the first 4 are used. The valid identifiers include:

##### DAS

Direct Access Spool data set block.

##### NITC

NJE node information table (\$NITs).

##### RECY

Recovery block.

#### LEN=

Specifies the length (in bytes) of a variable-length control block to be added to the change log record of the checkpoint data set. If the length is not specified here, the length value defaults to the length of this control block stored in the checkpoint information table (KIT). This keyword is ignored for fixed-length control blocks.

#### ADDR=

Specifies the address of the element to be checkpointed. The address **must** point to the beginning of a control block; it **must not** point to a field within that control block. If this parameter is omitted, only the header of the checkpoint area that is specified is checkpointed.

#### POST=

Specifies whether a post will occur for the checkpoint. The default is POST=NO, therefore, be certain that if your exit routine relies on a POST of the checkpoint (CKPT) processor you explicitly provide the \$CKPT POST=YES specification in that routine.

#### Note:

1. \$DOGJOE must be used to checkpoint updates to the \$JOE.
2. You must have control of the checkpoint data set when you issue this macro.
3. This macro returns a token value in R0 that uniquely identifies a particular checkpoint write. This token can be used as the CKPTID on the \$CHECK macro.
4. When you issue a \$CKPT under the command PCE, JES2 always posts the checkpoint processor. This causes JES2 to update the application copy of the checkpoint data set.

## **\$CPOOL**

### **Environment**

- Main task.
- \$WAIT cannot occur.

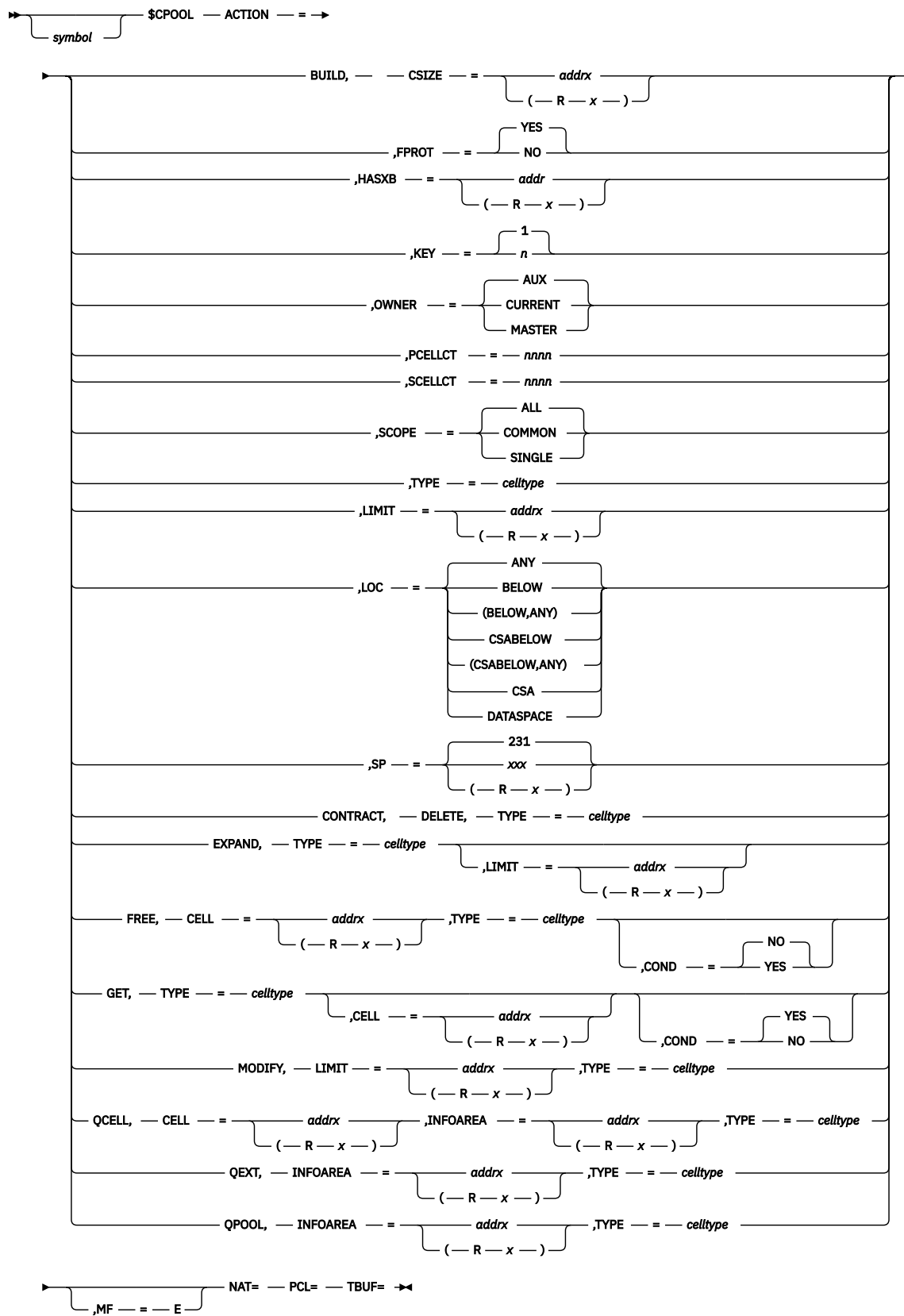
### **\$CPOOL – Build, delete, modify, or query a cell pool**

---

Use \$CPOOL to manage cell pool storage.

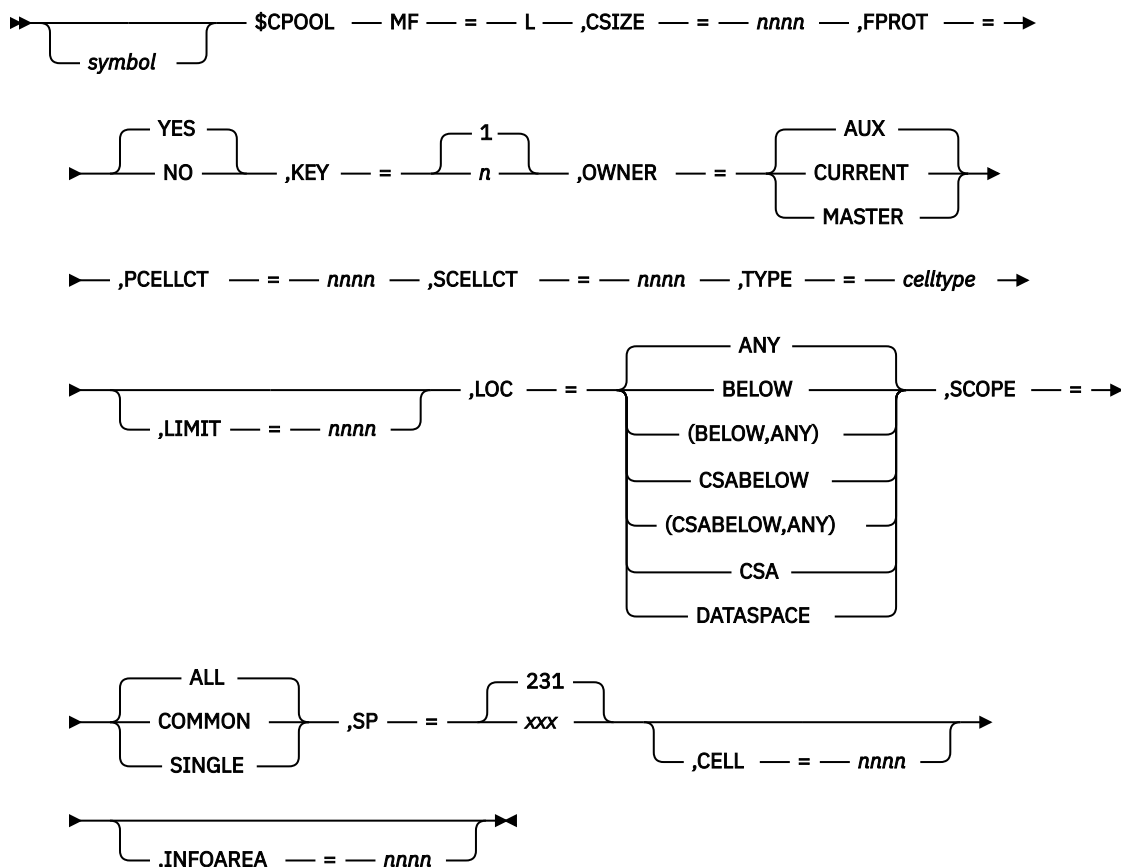


## Format description - Execute form



**Note:** If you specify a register (Rx), you can specify only registers R2 through R12.

### Format description - List form



**ACTION=**

Specifies the action to be performed. Depending upon which ACTION is specified, one or more of the subparameters might be required or optional.

For the execute form of the macro, this parameter is required. Do not code this parameter on the list form of the macro.

## BUILD

Creates a cell pool in a specified subpool by allocating storage and initializing the control information.

## CONTRACT

Frees storage for any unused extents in the cell pool.

**DELETE**

Deletes a previously built cell pool and frees storage and all cell pool control blocks.

**EXPAND**

Expands a cell pool by allocating storage for the extent.

**FREE**

Returns a cell to a cell pool. If the cell to be freed is in a data space, then the caller must be in AR ASC mode, and CELL= must be in a register with the corresponding access register set to access the data space.

**GET**

Obtains a cell from the previously built cell pool and returns its address to the location specified in CELL=. If there is no such cell pool, then the service uses the \$CPLTAB previously defined for this cell type. The cell is set to zero when obtained unless the pool is defined with CLEAR=NO.

## MODIFY

Modifies the limit or the number of cells in a cell pool.

**QCELL**

Queries information about a cell, and returns this information to a user-supplied work area.

**QEXT**

Queries information about an extent, and returns this information to a user-supplied work area.

**QPOOL**

Queries information about a cell pool, and returns this information to a user-supplied work area.

*Depending on which ACTION was coded, some of the following subparameters might be required or optional:*

**SP=xxx | (Rx) | 231**

Specifies the subpool from which \$CPOOL obtains the cell pool. If a register or variable is specified, the subpool number is taken from bits 24-31. If this parameter is not specified, the default is subpool 231. This parameter is allowed with the BUILD keyword.

**FPROT= YES | NO**

Specifies whether (YES) or not (NO) the data space is to be fetch protected.

If you specify FPRO=YES, you must also specify LOC=DATASPACE.

**OWNER= AUX | CURRENT | MASTER**

Specifies the owner of the data space as follows:

**AUX**

JES2 auxiliary address space

**CURRENT**

Current address space under which the subsystem is running and is only accessible from the creating address space.

**MASTER**

Address space 1.

If you specify OWNER=, you must also specify LOC=DATASPACE.

**SCOPE= ALL | COMMON | SINGLE**

Specifies the scope of accessibility of the data space as follows:

**ALL**

Any address space can access and connect to the data space.

**COMMON**

All address spaces can access the data space through a single ALET.

**SINGLE**

Only the owning address space can access the address space.

If you specify SCOPE=, you must also specify LOC=DATASPACE.

**TYPE=celltype**

Specifies the type of cell. Valid cell types are any of up to 8 alphanumeric characters, which are defined by the user.

The following cell types are reserved for use by IBM:

- BAT
- BSC
- B32K
- CB
- CDCT
- CDCTQS
- CDCTRNT
- CID
- CNIT

## **\$CPOOL**

- DLSJOB
- EDSMSG
- EVT
- GPQE
- HASP
- HEDR
- ICE
- IRE
- JQRB
- LMD
- NAT
- NHB
- NMAP
- NSA
- NTQ
- PAD
- PAGE
- PCL
- PP
- SAPID
- SCWA
- SCWADSP
- SJIO
- SMF
- SQD
- STAC
- TBUF
- TJEV
- TRE
- VTAM®
- WTO
- XCWELT
- XCWNODE
- XRQ
- ZGLMSG
- ZGLREQ
- ZJC

### **CELL=addrx | (Rx)**

Specifies the address or register in which the cell address is returned by the FREE, QCELL, and GET request. This parameter is required with the ACTION=FREE keyword.

### **COND=YES | NO**

With ACTION=GET or ACTION=FREE request, specifies whether (YES) or not (NO) to issue \$ERROR instead of passing a bad return code to the caller if cell pool request returns with non-zero return code.

This keyword is only valid when ACTION=GET or ACTION=FREE is specified. COND= defaults to YES if not specified for ACTION=GET and defaults to NO if not specified for ACTION=FREE or ACTION=GET.

**CSIZE= addrx | (Rx)**

Specifies the number of bytes of the cell requested. The minimum value of CSIZE is 1 byte.

**HASXB= addrx | (Rx)**

Specifies the address or register (R2-R15) that contains the address of the address space extension block (HASXB). If you do not specify HASXB=, JES2 obtains the HASXB from the address space block (HASB). If this is a common cell pool and there is no HASXB, then HASXB=NONE avoids using the HASXB address. HASXB=NONE is only valid on ACTION=GET, FREE, EXPAND, and query. This parameter is optional.

**INFOAREA= addrx | (Rx)**

Specifies the address or register containing the address of a user-supplied work area where the information is returned by QPOOL, QEXT, and QCELL. The information that is returned in this work area is mapped by different mapping macros, depending on which action was specified. If QEXT was specified, the work area is mapped by \$CPXWORK. If QPOOL was specified, the work area is mapped by \$CPPWORK. If QCELL was specified, the work area is mapped by \$CPCWORK.

**KEY=n | 1**

Specifies the storage key to be assigned to this subpool. The default is key 1. This parameter is required with the ACTION=FREE keyword.

**LIMIT= addrx | (Rx)**

Specifies the maximum number of cells in the cell pool.

**PCELLCT= nnnn**

Specifies the number of cells that are expected to be needed in the initial extent of the cell pool.

**SCCELLCT= nnnn**

Specifies the number of cells that are expected to be in each secondary or NON-INITIAL extent of the cell pool. The minimum is one cell.

**LOC= ANY | BELOW | (BELOW,ANY) | CSABELOW | (CSABELOW,ANY) | CSA | DATASPACE**

Specifies the location of virtual and real storage for the cell pool as follows:

**ANY**

Anywhere within private storage.

**BELOW**

Below 16 megabytes in private storage.

**(BELOW,ANY)**

Virtual storage is below the 16 megabyte line in private storage but it can be backed in real storage above the 16 megabyte line.

**CSABELOW**

Below 16 megabytes in CSA. (SP= must be specified as a CSA subpool).

**(CSABELOW,ANY)**

Virtual storage is below the 16 megabyte line in common storage but it can be backed in real storage above the 16 megabyte line. (SP= must be specified as a CSA subpool).

**CSA**

Anywhere in CSA. (SP= must be specified as a CSA subpool).

**DATASPACE**

In a data space. (The data space is named by appending the subsystem name with the *celltype* specified by TYPE=).

The following keywords are only valid if the cell pool is in a data space (LOC=DATASPACE):

**FPROT=YES**

Specifies whether the data space should be fetch protected (YES) or not (NO).

**OWNER=AUX**

Specifies who should own the data space.

## \$CPOOL

### MASTER

Address space 1.

### AUX

JES2 AUX address space.

### CURRENT

The current address space.

**Note:** OWNER=CURRENT data spaces are only accessible from the creating address space.

### SCOPE=ALL

Scope of accessibility of the data space.

### SINGLE

Only owning address space can access the data space.

### ALL

Any address space can connect (through ALESERV ADD) to the data space.

### COMMON

All address spaces can access the data space through a single ALET.

### MF=

Specifies the macro format, requesting an executable or list-form macro expansion.

### MF=L

Generates the list form.

### MF=E

Generates form to execute inline.

### MF=(E,label)

Generates a form to execute, using the list-form expansion defined at label 'label'.

## Return codes

The execute form of \$CPOOL provides the following decimal return codes in register 15. The valid return codes are described below, depending on which action was specified:

Action specified	Return codes	Description
BUILD	0	Processing successful
	4	Cell pool already exists
	8	No storage for cell pool
	12 and above	See the CSRBLD cell pool service in <a href="#">z/OS MVS Programming: Assembler Services Guide</a>
CONTRACT	0	Contraction was successful
	4	Contraction failed
DELETE	0	Processing successful
	8	No such cell pool
EXPAND	0	Processing successful
	4	LIMIT was exceeded
	8	No cells available, or CPOOL services not linked
	12 and above	See the CSRPEXP cell pool service in <a href="#">z/OS MVS Programming: Assembler Services Guide</a>

Action specified	Return codes	Description
FREE	0	Processing successful
	4	The last cell in an inactive extent was deallocated
	8	No such cell pool
	12 and above	See the CSRPFRE cell pool service in <a href="#">z/OS MVS Programming: Assembler Services Guide</a>
GET	0	Processing successful
	4	Could not get a cell, even though there is a cell pool (the reason could be either that the LIMIT was exceeded or there is not enough storage for another extent.)
	8	No cells available, or CPOOL services not linked
	12 and above	See the CSRPGET cell pool service in <a href="#">z/OS MVS Programming: Assembler Services Guide</a>
MODIFY	0	Processing successful
	4	LIMIT was increased by another caller, and now the new LIMIT is higher than the one requested by this issuance of the \$CPOOL macro.
	8	Invalid limit
QCELL	0	Processing successful
	8	CPOOL services not linked
	12 and above	See the CSRQCL cell pool service in <a href="#">z/OS MVS Programming: Assembler Services Guide</a>
QEXT	0	Processing successful
	8	No such cell pool
	12 and above	See the CSRQEX cell pool service in <a href="#">z/OS MVS Programming: Assembler Services Guide</a>
QPOOL	0	Processing successful
	8	No such cell pool
	12 and above	See the CSRQPL cell pool service in <a href="#">z/OS MVS Programming: Assembler Services Guide</a>

## Environment

- Execute and list form: JES2 Main Task, JES2 subtask (limited to initialization and termination), User, and functional subsystem (HASPFSM).
- \$WAIT cannot occur.
- Callers in AR ASC mode are supported.

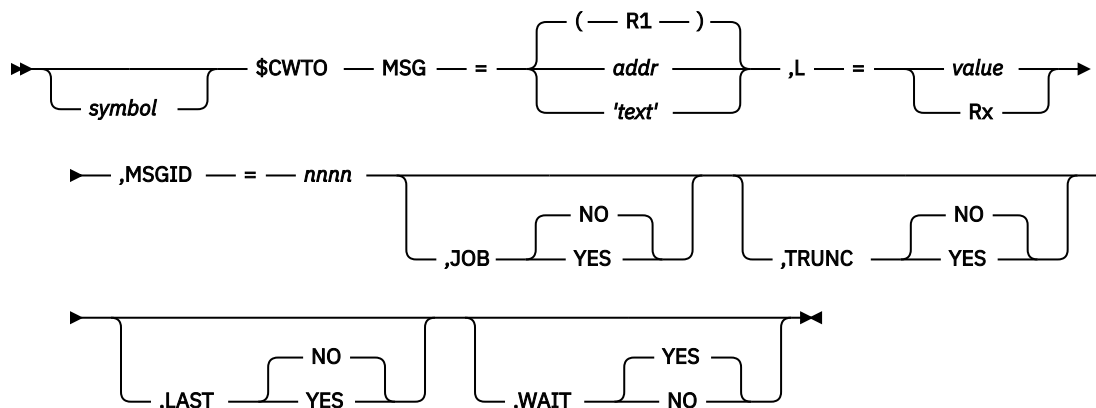
## \$CWTO – Command processor write to operator

Use \$CWTO to cause a write to operator to take place. This macro instruction returns control to the code issuing the macro. The command processor PCE must be in control when you issue this macro instruction. Note that, you cannot use \$CWTO in Exit 5 if the exit routine determines that JES2 should process the command. If you use this macro in Exit 5, your routine must do all required processing within the exit. When this processing is completed, your routine must notify HASPCOMM.

See Exit 5 documentation in [z/OS JES2 Installation Exits](#) for further details and recommendations.

Be sure to read “Usage notes” on page 91 below which describes the interaction of the L=, the JOB=, and the LAST= macro operands and considerations for when a console area is specified on a command processed by Exit 5.

## Format description



### MSG=

Specifies either the address of the text for the message or the text itself. If you specify the text, enclose the character string in single quotation marks ('). If you want the text to include single quotation marks, code two single quotation marks together. In addition, be sure to add the actual message length on the L= keyword.

### L=

Specifies either the length, in bytes, of the message text or a register containing the value. The length does not include the extra single quotation mark coded to allow the use of a single quotation mark within the text.

### MSGID=

Specifies a 3- or 4-digit decimal number, from 001-9999, to be written out with the message. See [z/OS JES2 Messages](#) for message ranges reserved for your use. You must include leading zeros.

### JOB=

Specifies whether the WTO is job related. Code JOB= as follows:

#### YES

The job name and number are inserted in the message. If you specify YES for a multi-line message, you must do so for every line of that message.

#### NO (default)

The message text remains unmodified.

### TRUNC=

#### YES

Any multiple-line WTO is truncated. Additional \$CWTO or \$CRET macro executions specifying message text result in the issuance of an MGCR or MGCRE. The MGCR or MGCRE treats the message text as a command to JES2.

#### NO (default)

No truncation takes place.

### LAST=

You must code LAST=YES on the last or only line of each message.

#### YES

Indicates that this is the last line of the multi-line write to operator (MLWTO) and begins a new line. LAST=YES is required for a single line WTO, or to signal the last line of MLWTO.



**NO (default)**

Indicates that one or more lines of a MLWTO will follow.

**WAIT=YES|NO**

Specifies the action JES2 is to take if there is no console message buffer (CMB) immediately available for this message. If JES2 is to wait until a CMB becomes available, code YES or omit this parameter. If you want JES2 to return without issuing the message and not wait, code NO.

If you code WAIT=YES or omit WAIT, exit 10, when it receives control, will be told that it can take an action that will result in a \$WAIT.

**Note:** The interaction of the L=, the JOB=, and the LAST= operands, and whether a console area was specified on the command itself, can cause unpredictable results and effects when Exit 5 receives control for a command. JES2 places the console area specification in field COMUCMA. If you do not specify a console area, JES2 sets this field to X'00'. If you do specify a console area, JES2 places the entire command response into a multi-line WTO and ignores any LAST=YES operands on the \$CWTO macro instruction.

## Usage notes

Use the following guidelines to ensure your command response messages are readable and are not truncated:

- If a console area is to be used, issue a control line first.

If a console area is to be used for a command response, JES2 processes the entire response as a multi-line WTO and ignores any LAST=YES operands on \$CWTO macro instructions. JES2 places the console area specification for a command in field COMUCMA. If there is no console area, JES2 sets this field to X'00'.

When a console area is to be used, your first \$CWTO is a control line for a multi-line WTO. Since JES2 places a message-id and time stamp at the beginning of a control line, there is only room for you to specify up to 16 characters on the MSG= operand. IBM suggests you model your message after the \$HASP636 message and echo the command in the available 16 characters. Do not code the JOB= or LAST= operands for a control line.

- Issue the remaining messages structuring your logic to reduce dependencies on whether an area was specified and to provide consistency when displaying job related messages. To do this:
  - Assume each single line and multi-line message will be issued independently as if an area weren't specified, that is, code LAST=YES on a \$CWTO for a "single" line message (Remember, it isn't really a single line if an area was specified causing LAST=YES to be ignored.) Similarly, for "multi-line" messages, code LAST=NO on the first and middle lines and LAST=YES on the "last" line.
  - If you code JOB=YES on a multi-line message, code it for each line of that message. For a single or multi-line message with JOB=YES, place the eight character JOBID followed by a blank in the first nine characters of the message text of the first or only message line. If an area wasn't specified, JES2 removes the JOBID from the message text, shifts the remaining text to the left, and issues a WTO with the specified JOBID. If you are issuing a multi-line message, place nine blanks at the beginning of text of the remaining lines.
  - Observe the following line length restrictions to reduce dependencies on whether an area was specified:
    - Place only the JOBID and jobname on the first line of a job related multi-line message, and not more than 25 characters on the first line of a non-job related multi-line message.
    - Limit the length of subsequent message lines to 70 characters if JOB=NO, or 61 characters if JOB=YES.

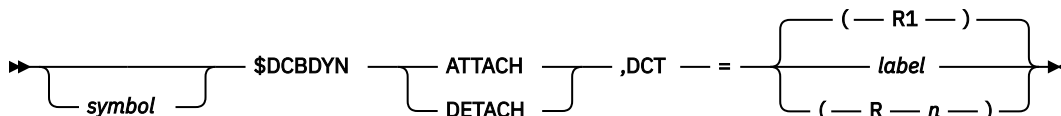
## Environment

- Main task.
- \$WAIT can occur.

## \$DCBDYN – Call the dynamic DCB service routine

Use \$DCBDYN macro as the interface to the \$DCBDYN service routine to attach or detach a JES2 data control block (DCB) and data extent block (DEB) for a specified device control table (DCT).

### Format description



#### ATTACH

Requests that a DCB and/or DEB (if one is required) be dynamically created. If the DCT does not require a DCB (for example, one has already been created), JES2 takes no further action.

#### DETACH

Requests that the specified DCB/DEB be dynamically deleted. If the DCT does not require a DCB or if there is no DCB already attached, JES2 takes no further action.

#### DCT=

Specifies a label or register containing the address of the DCT to either be attached or detached.

### Return codes

The following return codes (in decimal) are returned in register 15.

#### Return Code Meaning

- |          |  |
|----------|--|
| <b>0</b> | DCB successfully ATTACHed or DETACHed as requested |
| <b>4</b> | DCB ATTACH failed (GETMAIN unsuccessful)           |

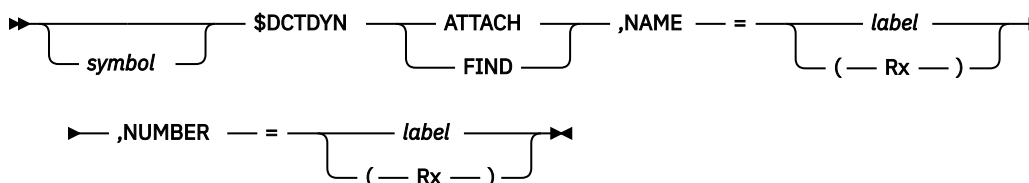
### Environment

- Main task.
- 31-bit addressing mode only.
- \$WAIT can occur.

## \$DCTDYN – Call the dynamic DCT service routine

Use \$DCTDYN macro as the interface to the \$DCTDYN service routine to attach or find a JES2 device control table (DCT). This macro passes the DCT name and subscript and type of request to the calling routine.

### Format description



#### action

Specifies the action requested.

**ATTACH**

Requests that the specified DCT be located, or if it doesn't exist, that a new one should be created. ATTACH is only a valid specification if this macro is called by JES2.

**FIND**

Requests that the specified DCT be located. If the DCT is successfully located, its address is returned in Register 1.

**NAME=**

Specifies the address of an 8-byte field that contains the name of the specified DCT. NAME can be specified as a register (1 to 12) or the name of the field containing the address of the DCT name. The address is loaded into register 1.

**NUMBER=**

Specifies the subscript (the binary value) of the DCT. NUMBER can be a register (0, 2 to 12) or the name of a field containing the subscript. The value is loaded into register 0. \$DCTDYN supports devices that have only one subscript, such as lines and local printers. It does not support devices that have multiple subscripts, such as remote printers.

## Return codes

The following return codes (in decimal) are returned in register 15.

Return Code	Meaning
0	DCT successfully found for either FIND or ATTACH request
4	DCT successfully ATTACHed if ATTACH requested
8	DCT not found – ATTACH not specified
12	DCT FIND/ATTACH not successful. The subscript specified by NUMBER= was either: not within the valid range, required and not specified, or not required and specified.
16	DCT ATTACH not successful – error in \$GETMAIN
20	DCT FIND/ATTACH not successful – DCT table not found
24	DCT FIND/ATTACH not successful – UCT not found

## Environment

- Main task.
- \$WAIT cannot occur.

## \$DCTTAB – Map DCT table entries

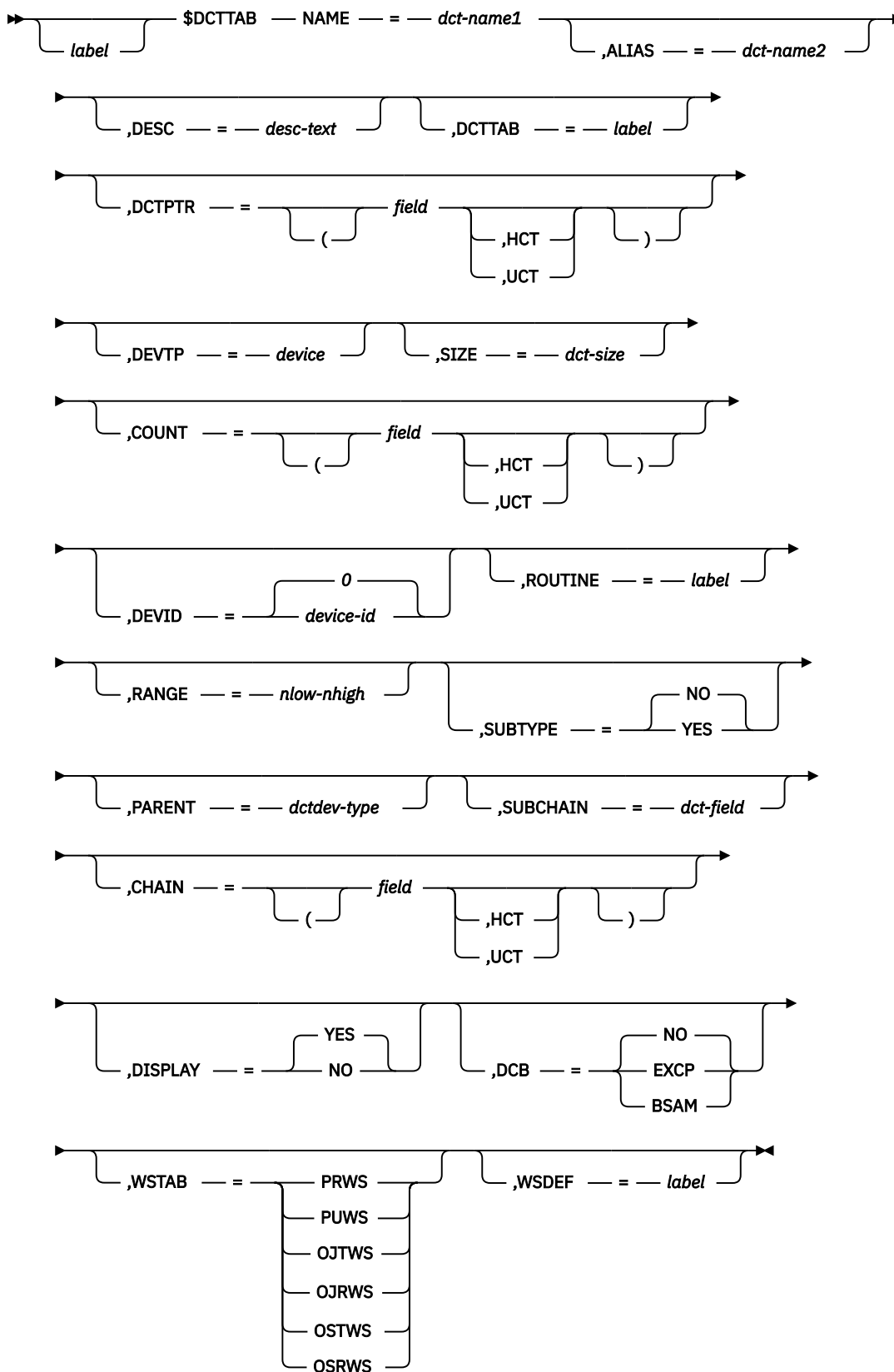
Use \$DCTTAB to map and generate DCT table entries.

\$DCTTAB entries are used to define the start of a user table (\$DCTTAB TABLE=USER...) or a JES2 table (\$DCTTAB TABLE=HASP...), the end of a table (\$DCTTAB TABLE=END) or an entry in a table (\$DCTTAB NAME=CALS ...).

**Note:** The format description that follows breaks the macro into a **boundary form** (the form that starts or ends a table) and an **entry form** (the form that defines each table entry).



## Entry form



### TABLE=

Specifies the start (TABLE=HASP) and end (TABLE=END) of a DCT table. If neither this keyword nor NAME= is specified, JES2 generates the DTAB DSECT.

**HASP**

Specifies that this is a HASP table.

**NOENTRY**

Specifies that an ENTRY statement need not be generated for the label of this DCT table.

**USER**

Specifies that this is a USER table.

**DYNAMIC**

Specifies that this is a DYNAMIC table.

**END**

Specifies the end of the DCT table.

**Note:** If TABLE= is specified, all other keywords on this macro are ignored.

**NAME=**

Specifies a 1- to 8-character DCT name for this DCT type.

**ALIAS=**

Specifies a 1- to 8-character DCT name to be used as an alternate

**DESC=**

Specifies a 1- to 24-character description of this DCT type. Blanks are allowed if the text is enclosed in single quotation marks. This keyword is used for documentation purposes only.

**PCETAB=**

Specifies the label on the PCE table entry in the same assembly module that corresponds to this DCT type. This keyword causes this DCT to be defined in a one-to-one PCE-DCT correspondence.

**Note:** PCETAB= and PCEPTR= are mutually exclusive.

**PCEPTR=**

Specifies the name of a fullword field that contains the address of the PCE that handles DCTs of this type when not organized in a one-to-one PCE-DCT correspondence as specified by PCETAB=.

**Note:** PCETAB= and PCEPTR= are mutually exclusive.

**field**

Specifies an HCT field if this is a HASP table and a UCT field if this is a USER table.

**(field,HCT)**

- Indicates a field in the HCT.

**(field,UCT)**

Indicates a field in the UCT.

**(label,ADDR)**

Indicates a label in the current module.

**(offset,TOKEN)**

Indicates a field in the block pointed to by the token specified by the PCEPTRTK= parameter.

**TOKEN**

Indicates a field in the block pointed to by the token specified by the PCEPTRTK= parameter .

**PCEPTRTK=**

Specifies the NAME associated with a name/token pair (created using the IEANTCR callable service) which contains the address of the control block that contains the PCE pointer. NAMES can be up to 16 bytes long (and must match the name specified on a IEANTCR call). The first 4 bytes of the token are assumed to be the control block address. The remaining 12 bytes of the token are not used.

The control block address (from the token) is combined with the offset specified on PCEPTR= to determine the PCEPTR field. PCEPTRTK= is required, and only allowed, if you specify PCEPTR=(offset,TOKEN).

NAME/TOKEN pairs can be created at various levels. JES2 supports tokens at the SYSTEM, HOME, or TASK level. The second operand on PCEPTRTK= specifies the level passed on the IEANTRT service call.

**PCEPTRTK=(name,SYSTEM)**

Indicates SYSTEM level

**PCEPTRTK=(name,SUBSYS)**

Indicates SYSTEM level with the last 4 bytes of the 16-byte name replaced by the subsystem id.

**PCEPTRTK=(name,HOME)**

Indicates HOME level

**PCEPTRTK=(name,PRIMARY)**

Indicates PRIMARY

**PCEPTRTK=(name,TASK)**

Indicates TASK level

**PCEPTRTK=name**

Defaults to TASK level

**DEVTP=**

Specifies a unique value used for the 1-byte DCTDEVTP field that defines this DCT type. This is a required keyword.

**WSTAB=**

Specifies the type of work selection table that corresponds to this DCT.

**PRWS**

Indicates that this is a printer work selection table.

**PUWS**

Indicates that this is a punch work selection table.

**OJTWS**

Indicates that this is an offload job transmitter (OFFn.JT) work selection table.

**OJRWS**

Indicates that this is an offload job receiver (OFFn.JR) work selection table.

**OSTWS**

Indicates that this is an offload SYSOUT transmitter (OFFn.ST) work selection table.

**OSRWS**

Indicates that this is an offload SYSOUT receiver (OFFn.SR) work selection table, a LJTWWS for line job transmitter (Ln.JTn) or LSTWS for line SYSOUT transmitter (Ln.STn).

**Note:**

1. This keyword is required for DCTs that support work selection.
2. If this keyword specification is other than those listed above, JES2 assumes that the table type is user defined.
3. If WSTAB is specified, you must also specify WSDEF=.

**WSDEF=**

Specifies the address of the default work selection list for this device.

**SIZE=**

Specifies the size of this DCT type. This can be specified either as an equated symbol or computed as SIZE – DCT.

**CHAIN=**

Specifies the name of a fullword field from which all DCTs of this type are to be chained.

**field**

Specifies an HCT field if this is a HASP table and a UCT field if this is a USER table.

**(field,HCT)**

- Indicates a field in the HCT.

**(field,UCT)**

Indicates a field in the UCT.

**(label,ADDR)**

Indicates a label in the current module.

**(offset,TOKEN)**

Indicates a field in the block pointed to by the token specified by the CHAINTK= parameter.

**TOKEN**

Indicates a field in the block pointed to by the token specified by the CHAINTK= parameter.

**CHAINTK=**

Specifies the NAME associated with a name/token pair (created using the IEANTCR callable service) which contains the address of the control block that contains the CHAIN pointer. NAMES can be up to 16 bytes long (and must match the name specified on a IEANTCR call). The first 4 bytes of the token are assumed to be the control block address. The remaining 12 bytes of the token are not used.

The control block address (from the token) is combined with the offset specified on CHAIN= to determine the chaining field. CHAINTK= is required, and only allowed, if CHAIN=(offset,TOKEN) was specified.

NAME/TOKEN pairs can be created at various levels. JES2 supports tokens at the SYSTEM, HOME, or TASK level. The second operand on CHAINTK= specifies the level passed on the IEANTRT service call.

**CHAINTK=(name,SYSTEM)**

Indicates SYSTEM level

**CHAINTK=(name,SUBSYS)**

Indicates SYSTEM level with the last 4 bytes of the 16-byte name replaced by the subsystem id.

**CHAINTK=(name,HOME)**

Indicates HOME level

**CHAINTK=(name,PRIMARY)**

Indicates PRIMARY

**CHAINTK=(name,TASK)**

Indicates TASK level

**CHAINTK=name**

Defaults to TASK level

**COUNT=**

Specifies the name of a fullword field that contains the number of DCTs defined for this DCT type.

**field**

Specifies an HCT field if this is a HASP table and a UCT field if this is a USER table.

**(field,HCT)**

- Indicates a field in the HCT.

**(field,UCT)**

Indicates a field in the UCT.

**(label,ADDR)**

Indicates a label in the current module.

**(offset,TOKEN)**

Indicates a field in the block pointed to by the token specified by the COUNTTK= parameter.

**TOKEN**

Indicates a field in the block pointed to by the token specified by the COUNTTK= parameter.

**COUNTTK=**

Specifies the NAME associated with a name/token pair (created using the IEANTCR callable service) which contains the address of the control block that contains the CHAIN pointer. NAMES can be up to 16 bytes long (and must match the name specified on a IEANTCR call). The first 4 bytes of the token are assumed to be the control block address. The remaining 12 bytes of the token are not used.

The control block address (from the token) is combined with the offset specified on COUNT= to determine the chaining field. COUNTTK= is required, and only allowed, if COUNT=(offset,TOKEN) was specified.



NAME/TOKEN pairs can be created at various levels. JES2 supports tokens at the SYSTEM, HOME, or TASK level. The second operand on COUNTTK= specifies the level passed on the IEANTRT service call.

**COUNTTK=(name,SYSTEM)**

Indicates SYSTEM level

**COUNTTK=(name,SUBSYS)**

Indicates SYSTEM level with the last 4 bytes of the 16-byte name replaced by the subsystem id.

**COUNTTK=(name,HOME)**

Indicates HOME level

**COUNTTK=(name,PRIMARY)**

Indicates PRIMARY

**COUNTTK=(name,TASK)**

Indicates TASK level

**COUNTTK=name**

Defaults to TASK level

**DEVID=**

Specifies the device ID that is placed into the first byte of the DCTDEVID field. If a device does not have a device ID, this field is set to 0.

**ROUTINE=**

Specifies the name of the routine used to initialize the DCTs.

**RANGE=**

Specifies the lower (nlow) and upper (nhigh) range limits of the subscript values that are allowed for this DCT type. If this keyword is not specified, the DCTs will not contain subscripts.

**SUBTYPE=**

Specifies whether this DCT has other DCTs chained off it within a subchain. The default is NO.

**PARENT=**

Specifies the DCTDEVTP of the DCT off which this DCT is chained.

**Note:** If this keyword is specified, SUBCHAIN= must also be specified.

**SUBCHAIN=**

Specifies the name of the field in this DCT which chains the DCT off the parent DCT and any other DCT types within the subchain.

**DISPLAY=**

Specifies whether (YES) or not (NO) this DCT will be displayed by a \$D U operator command.

**YES**

Indicates that this DCT is chained within the DCTPOOL chain and therefore displayed by the \$D U operator command. This is the default.

**NO**

Indicates that this DCT is chained within the DCTPOL2 chain and therefore not displayed by the \$D U operator command.

**DCB=**

Specifies whether either a DCB or DEB is built for this DCT.

**EXCP**

Indicates that both an EXCP DCB and DEB be built.

**BSAM**

Indicates that a BSAM DCB be built.

**NO**

Indicates that neither a DCB nor a DEB be built for this DCT. This is the default.

## **Environment**

- JES2 main task or during initialization and termination.
- \$WAIT is not applicable – this macro generates a DSECT or a static table entry; it does not generate executable code.

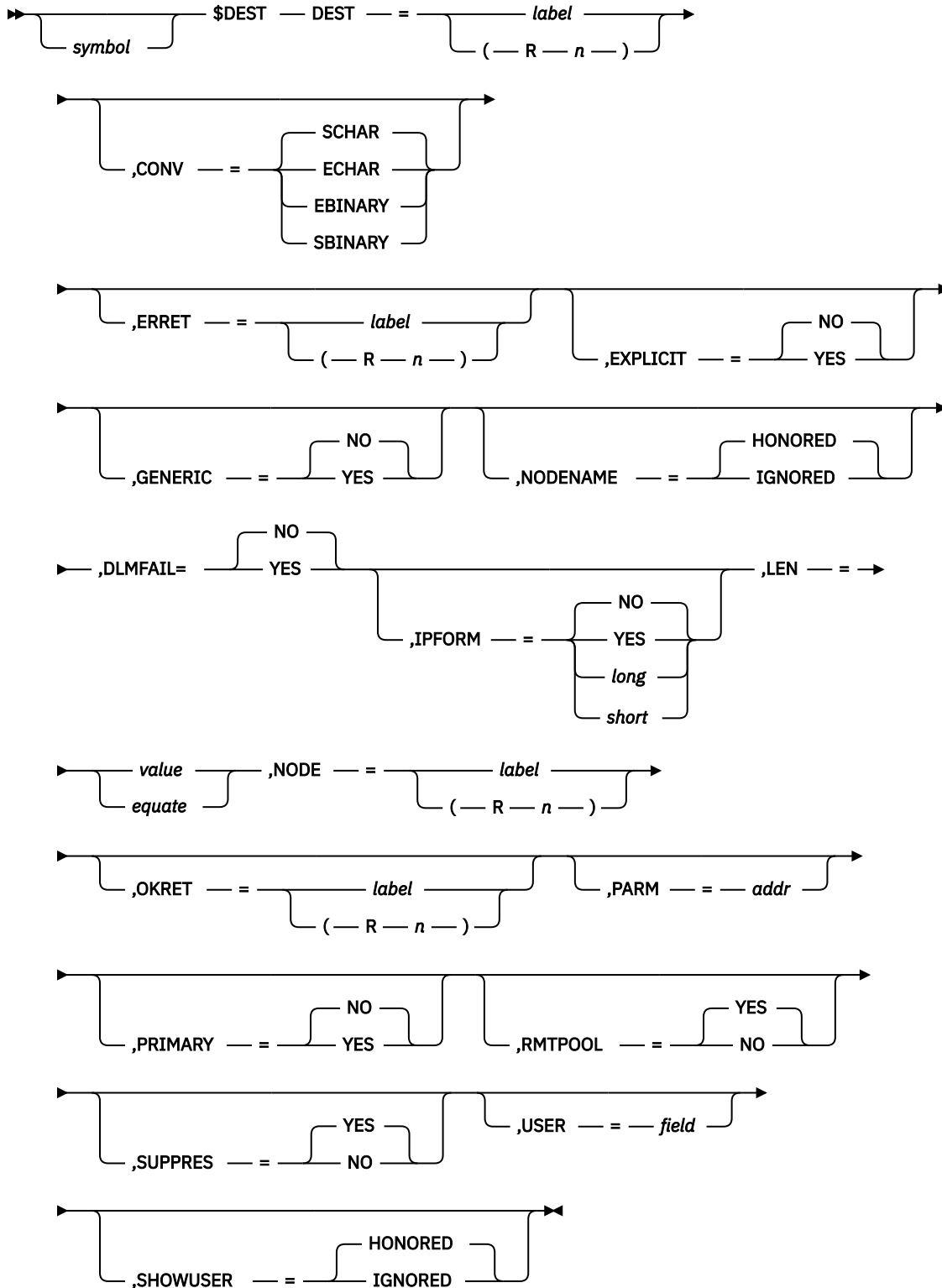
## **\$DEST – Convert symbolic destinations and binary route codes**

---

Use \$DEST to convert a symbolic destination to a binary route code or a binary route code to a symbolic destination.

**Note:** In all examples cited below, node 1 refers to the local node.

## Format description



### DEST=

Specifies the address of the symbolic destination for which a binary route code is obtained or the address of an area that contains a binary route code which is to be replaced by a symbolic destination. If register notation is used, the destination address must be loaded into the designated register before the execution of this macro instruction.

## **\$DEST**

**Note:** If CONV=EBINARY was specified, the minimum length required for the DEST field is 16 bytes. If CONV=SBINARY, the length of the DEST field must be at least 17 bytes, unless the destination is at the local node and SUPPRES=YES was specified, in which case the minimum length is 16.

### **,CONV=**

Specifies the type of conversion that is required as follows:

#### **SCHAR**

The symbolic destination specified by DEST= is to be converted to a binary value and returned in R1. The DEST value can be explicit (for example, LOCAL or NnnnnRnnnn) or a DESTID value.

#### **ECHAR**

The symbolic destination specified by DEST= is to be converted to a binary value and returned in R1. The DEST value **must** be explicit (for example, NnnnnRnnnn).

#### **SBINARY**

Specifies a binary route code is to be converted to a symbolic destination. If only a single DESTID matches, it is converted and returned. If no DESTIDs or more than one DESTID matches, then the explicit form (for example, NnnnnRnnnn or LOCAL) is returned. The DEST field must be at least 17 bytes in length, unless the destination is at the local node, and SUPPRES=YES was specified, in which case the minimum length is 16 bytes.

#### **EBINARY**

Specifies a binary route code is to be converted to a symbolic destination. Only the explicit forms are checked (for example, NnnnnRnnnn or LOCAL) and returned. The DEST field must be at least 16 bytes in length.

### **ERRET=**

Specifies a location to which control is returned if the specified destination is not valid.

### **LEN=**

Specifies the length or an equated value of the length of the symbolic destination.

### **NODE=**

Specifies the address or a register that contains the address of a halfword field that contains the default node number used to construct the binary route code. If register notation is used, the node number is loaded into the designated register before the execution of this macro instruction. NODE= is required if you code CONV=SCHAR or CONV=ECHAR.

### **EXPLICIT =**

Specifies whether (YES) or not (NO) this \$DEST macro call ignores the DESTDEF initialization statement that can affect how job and SYSOUT destinations are processed. EXPLICIT=NO is the default.

If you specify EXPLICIT=YES, the Ndest=, R|RM|RMTdest=, and Udest= parameters on the DESTDEF initialization statement provide only their default values on this \$DEST macro call.

### **GENERIC=**

Specifies whether or not generic userids can be used.

#### **YES**

Specifies that the last character of a userid can be an asterisk (\*).

#### **NO**

Specifies that an \* is not a valid character in a userid.

This parameter is only valid when CONV=SCHAR and a userid field are specified.

### **NODENAME=**

Specifies whether or not DESTDEF NODENAME will affect \$DEST processing.

**Note:** This parameter is only valid when CONV= SCHAR or ECHAR is specified.

#### **HONORED**

The setting of DESTDEF NODENAME will affect \$DEST processing. If DESTDEF NODENAME=REQUIRED, then the destination must be either a valid destid or a userid explicitly

prefixed with a node. If DESTDEF NODENAME=OPTIONAL, then userids do not require explicit node qualification.

**IGNORED**

The setting of DESTDEF NODENAME will not affect \$DEST processing.

The default value for DESTDEF NODENAME is HONORED.

**DLMFAIL=**

Determines whether delimiters within the destination are to be honored (NO) or considered an error (YES). If yes is specified, the destination passed in must be followed by trailing blanks or nulls. The default is DLMFAIL=NO.

**IPFORM=**

This parameter determines what form, if any IP-format destination is valid.

**Note:** This parameter is only valid when CONV= SCHAR or ECHAR is specified.

**IPFORM=NO (Default)**

Specifies that JES2 allows no form of IP-format destination.

**IPFORM=YES**

Specifies that the input destination can be in IP format, for example: 'node.IP:-address' or 'IP:ip-address'. You must also specify USER=.

**IPFORM=LONG**

Specifies that the input destination can be in IP format, for example: 'node.IP:-address' or 'IP:ip-address'. You must also specify USER=.

**IPFORM=SHORT**

Specifies that input destination of <IP> is allowed. You must also specify USER=.

**OKRET=**

Specifies the address of the routine that is to receive control when JES2 passes back a zero return code.

**PARM=**

Specifies the address of storage to be used for a parameter list for the \$DEST service, which will be filled in automatically by the macro. The default for this parameter is 12(R13), that is the address contained in register 13 plus 12 bytes.

**PRIMARY=****YES**

Causes JES2 to return the primary DESTID instead of an explicit route code if there are multiple route codes which match the binary input. This parameter takes effect only when you code CONV=SBINARY.

**NO**

Causes CONV=SBINARY to work as described above.

**RMTPOOL=****YES**

Specifies that JES2 should change the destination to reflect remote pooling.

**NO**

Remote pooling not in effect.

**SUPPRES =**

Specifies whether or not destinations at the local node are converted and displayed differently from destinations at other nodes.

**YES**

Specifies that a first-level destination of the local node name is not included in the final destination.

## **\$DESTDYN**

When converting from binary input to character at node 1, a destination of 0001 0005 is resolved as R5. However, a destination of 0002 0005 is resolved as N2.R5 because N2 is not the local node.

When converting from character input to binary at node 1, a destination of N1.U5 will resolve to a destination of 0000 0005, with no userid placed in the area specified by USER=.

When converting from character input to binary at node 1, N2.U5 resolves to 0002 0000 with the characters 'U5' placed in the area specified by USER=, because N2 is not the local node.

SUPPRES=YES is the default.

### **NO**

Specifies that a first-level destination is included in the final destination.

When converting from binary input to character, a destination of 0001 0005 resolves as N1.R5.

When converting from character input to binary, a destination of N1.U5 resolves to a binary destination of 0001 0000 with the characters 'U5' placed in the area specified by USER=.

### **USER=**

If you code CONV=SBINARY or CONV=EBINARY, you can supply a userid for the field to be placed in the output area.

If you code CONV=SCCHAR or CONV=ECCHAR, the conversion routine will separate the userid from the route code, and place the userid in the field.

### **SHOWUSER=**

Specifies whether or not DESTDEF SHOWUSER will be honored during \$DEST processing.

#### **HONORED**

The setting of DESTDEF SHOWUSER will be honored during \$DEST processing

#### **IGNORED**

The setting of DESTDEF SHOWUSER will be ignored during \$DEST processing.

The default value for DESTDEF SHOWUSER is HONORED.

## **Return codes**

The following return codes (in decimal) are returned in register 15.

### **Return Code**

#### **Meaning**

**0**

Specified destination valid and converted

**4**

Destination is not valid – error return

## **Environment**

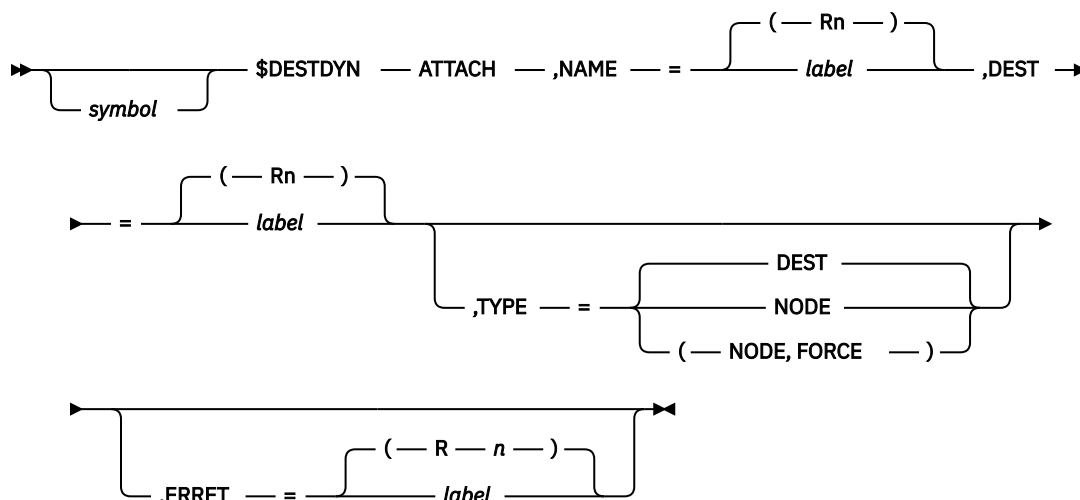
- All environments.
- MVS WAIT or \$WAIT cannot occur.
- Callers in AR ASC mode are supported. However, all data areas passed must be addressable in primary ASC mode.

## **\$DESTDYN – Attach a JES2 DESTID**

---

Use \$DESTDYN to generate a destination identifier (DESTID) or alter an existing DESTID.

## Format description



### ATTACH

Specifies to either generate a DESTID or alter an existing DESTID. This positional keyword is required.

### NAME=

Specifies the address of a field or a register (R1-R12) containing the address of the field that contains an 8-byte DESTID.

### DEST=

Specifies a label or a register (R0 or R2-R12) containing the address of a destination value to be assigned to the destination name pointed to by NAME=. If TYPE=DEST, this destination value is a symbolic destination of \$MAXRCLN bytes in length. (\$MAXRCLN is currently defined by 10 bytes.) If TYPE=NODE, this destination value is a binary destination value.

### TYPE=

Specifies the type of DESTID to be defined and the type of checking required.

#### DEST

Indicates that the destination ID is a destination name. If the destination name is already defined as a DESTID, it can be altered as required; if it is a node, alteration is not allowed.

#### NODE

Indicates that the destination ID is a node name. If the destination node name is already defined as a DESTID, it must be for the same node. If the destination is not previously defined, it is changed to indicate it is a destination only.

#### (NODE, FORCE)

The destination is a node name. The definition previously defined as a node is changed to indicate that it is a dest only. If the current name is already a destid or a nodename, the definition is forced to be a nodename regardless of the current value of the destination.

### ERRET=

Specifies a label or a register containing the address of the routine that receives control if an error occurs during \$DESTDYN processing.

## Return codes

The following return codes (in decimal) are returned in register 15.

### Return Code

#### Meaning

#### 0

Processing successful. Register 0 contains the binary destination matching the input destination. Register 1 contains the RDT (remote destination table) address.

## **\$DILBERT**

**4**

GETMAIN error for a new RDB (remote destination table block)

**8**

\$DEST returned an error for the destination value passed.

**12**

\$DEST indicated that the destination value passed is an explicit destination route value. Register 0 contains the binary destination matching the input destination. Register 1 contains the binary destination matching the destination name.

**16**

\$DEST has detected an invalid alteration attempt of an existing RDT. Register 0 contains the binary destination matching the input destination.

## **Environment**

- Main task.
- \$WAIT cannot occur.

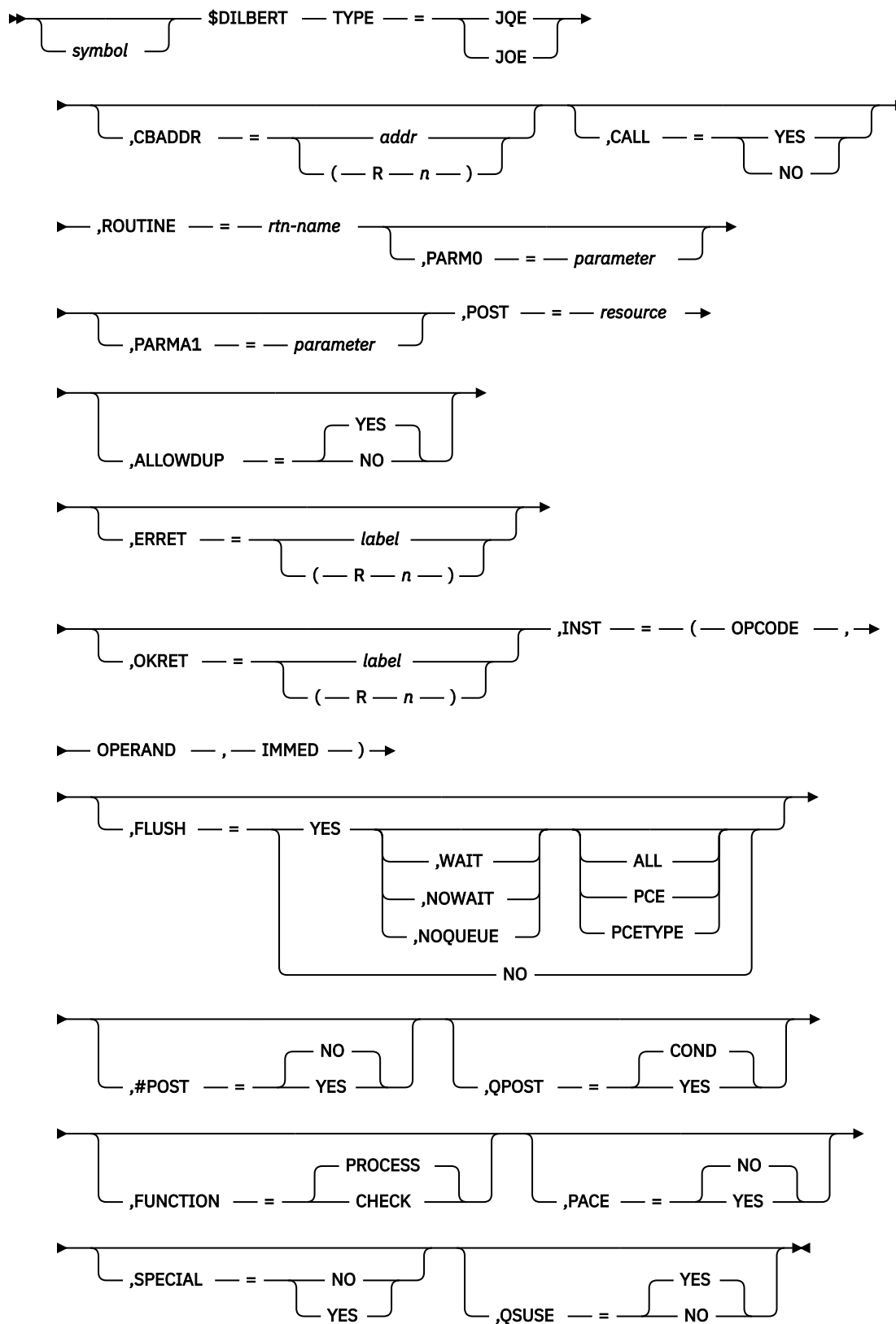
## **\$DILBERT – Do it later BERT services**

---

Use \$DILBERT to specify a routine to gain control when JES2 releases the block extension reuse table (BERT) lock for a specific job.



## Format description



### TYPE=

Specifies the type of BERT. The supported types are JQE and JOE.

**CBADDR=**

Specifies the control block JES2 passes to the routine in register 1. This control block might be at a different address once JES2 passes it to the specified routine. It will however represent the same data within the checkpoint.

**CALL=**

Specifies whether (YES) or not (NO) JES2 will immediately call the routine specified on the ROUTINE= parameter if the BERT lock is currently available. CALL=YES is mutually exclusive with INST and FLUSH.

**ROUTINE=**

Specifies the name of the routine to get control after JES2 frees the BERT lock for the job. ROUTINE is mutually exclusive with INST= and FLUSH=.

**PARMO=**

An optional parameter JES2 passes in register 0 to the routine you specified by ROUTINE=.

**PARMA1=**

An optional parameter to be passed to the \$DILBERT service in access register 1.

**POST=**

Specifies the resource to be \$POSTed when the queue element becomes available. This specification automatically causes suppression of duplicate DWAs, that is, as if ALLOWDUP=NO were specified.

**ALLOWDUP=**

Specifies whether (YES) or not (NO) this dilbert work area (DWA) should be queued if it is a duplication of an already queued DWA. The default is YES.

**ERRET=**

Specifies a label to be branched to or a register to be branched on if a non-zero return code is returned in R15.

**OKRET=**

Specifies a label to be branched to or a register to be branched on if a zero return code is returned in R15.

**INST=**

Specifies a single triplet that specifies an operation to be performed against a field (OPERAND) within CBADDR with an immediate type instruction.

**OPCODE**

The operation instruction can be one of the following: MVI (move), XI (exclusive OR), OI (OR), or NI (AND).

**OPERAND**

The field name within CBADDR that JES2 will manipulate as instructed by the opcode and immediate operand.

**IMMED**

The 'immediate' data field the opcode uses to manipulate the operand.

For example,

```
INST=(OI,JQEFLAG7,JQE7SPIN)
```

JES2 manipulates JQEFLAG7 based on an OR operation against field JQE7SPIN.

INST= is mutually exclusive with ROUTINE=, CALL=YES, and FLUSH=.

**FLUSH=**

Specifies whether (YES) or not (NO) the pending work for the queue element should be executed by JES2. If YES is specified, an optional second operand of WAIT/NOWAIT/NOQUEUE can be specified.

**WAIT**

The PCE is \$WAITed until all pending work is finished.

**NOWAIT**

If the BERT lock for the job is not available, a return code of 4 is returned and the PCE is \$POSTed for WORK when the flush completes.

**NOQUEUE**

If the BERT lock for the job is not available, a return code of 4 is returned and the PCE is not \$POSTed when the flush completes.

Additionally, an optional third operand of ALL/PCE/PCETYPE can be specified with FLUSH=YES.

**ALL**

All work for this JQA/JOA should be flushed.

**PCE**

Only work queued by the current PCE should be flushed.

**PCETYPE**

Only work queued by PCEs of the same type as the current PCE should be flushed.

FLUSH= is mutually exclusive with INST=, CALL=, and ROUTINE=.

**QPOST=**

Specifies how to call \$DOGJQE when a JQE is returned. QPOST=YES means that a \$DOGJQE QPOST=YES will be used; QPOST=COND means that a \$DOGJQE QPOST=COND will be used. The default is COND.

**Note:** The QPOST parameter is not valid if TYPE=JOE is specified.

**#POST=**

Specifies how to call \$DOGJQE or \$DOGJOE when a JQE or JOE is returned. #POST=YES means that a \$DOGJQE or \$DOGJOE #POST=YES will be used; #POST=NO means that a \$DOGJQE or \$DOGJOE #POST=NO will be used. The default is NO.

**FUNCTION=**

Specifies the function (PROCESS or CHECK) that \$DILBERT will perform.

**PROCESS**

This will allow updates to BERTs to be deferred until the BERT lock is no longer held by another processor. FUNCTION=PROCESS is the default.

**CHECK**

This will examine the pending \$DILBERTed work to find a DWA with the same routine and parameters. A return code of 4 indicates that a matching DWA was found. A return code of 0 indicates no matching DWA was found. ROUTINE= is a required parameter, while PARM0= and PARMA1= are optional. Even if PARM0 or PARMA1 is not specified, its default of zero will be used when comparing to find a match.

For example:

```
FUNCTION=CHECK, ROUTINE=ABCD
```

JES2 examines the pending DWAs to search for the one that is under the following conditions:

- ROUTINE=ABCD is specified.
- PARM0 or PARMA1 is set to its default.

FUNCTION=CHECK is mutually exclusive with TYPE=, CBADDR=, CALL=, POST=, ALLOWDUP=, PACE=, INST=, FLUSH=, QPOST=, QSUSE= and #POST=.

**PACE=**

Specifies whether (YES) or not (NO) pacing should be done on behalf of requests calling the routine supplied. The default is NO.

If pacing is requested, the DILBERT PCEs dispatch no more than one DWA with the same entry point value as the routine specified in ROUTINE=.

You must specify a routine address if PACE=YES is requested. PACE=YES and CALL=YES are mutually exclusive.

## **\$DISTERR**

### **SPECIAL=**

Specifies whether (YES) or not (NO) the caller wants to skip getting the BERT lock for the JQA/JOA. The default is YES if INST= or POST= is specified. Otherwise the default is NO.

### **QSUSE=**

Specifies whether (YES) or not (NO) the queues are guaranteed to be owned if the routine named by ROUTINE= is called. The default is YES.

## **Environment**

- JES2 main task only.
- \$WAIT cannot occur.

## **Return codes**

Return codes (R15 on exit) FUNCTION=PROCESS

### **Return Code**

#### **Meaning**

**0**

Processing successful (no errors).

**4**

FLUSH request with WAIT=NO and BERT lock not available; or any other duplicate request and ALLOWDUP=NO specified or implied.

**8**

DWA constructed, request deferred.

Return codes (R15 on exit) FUNCTION=CHECK

### **Return Code**

#### **Meaning**

**0**

NO matching DWA found.

**4**

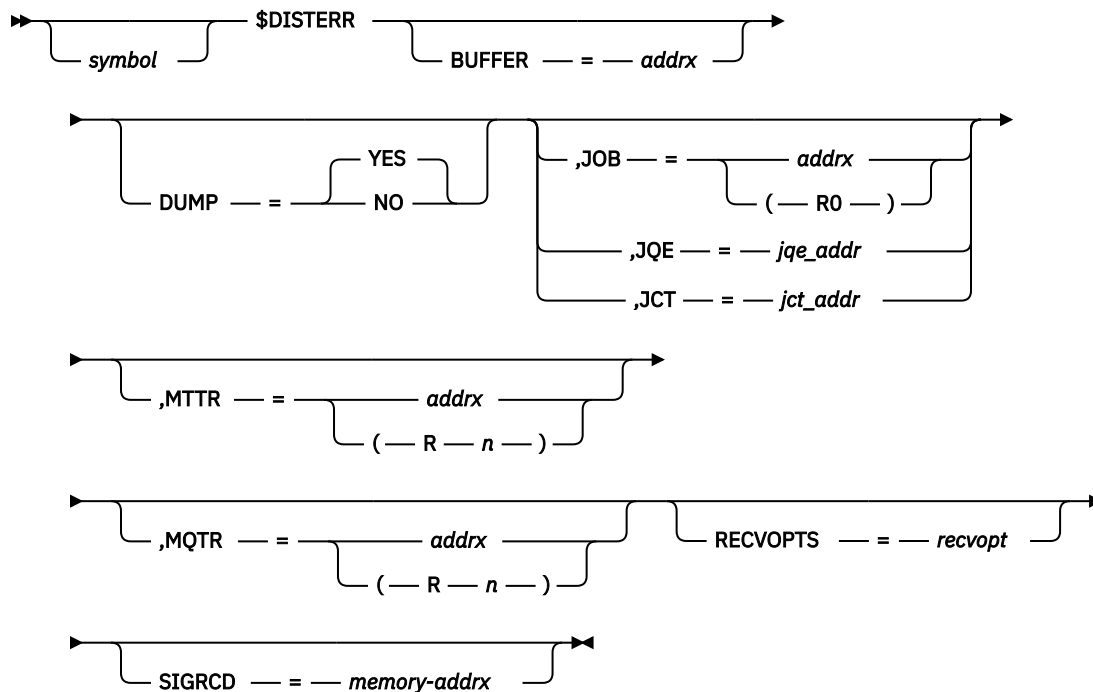
Matching DWA found.

## **\$DISTERR – Indicate disastrous error**

---

Use \$DISTERR to indicate that a disastrous error has occurred. The macro instruction causes the message \$HASP096 DISASTROUS ERROR AT SYMBOL symbol IN CSECT module to be printed out on the \$ERR and \$LOG consoles.

## Format description



### symbol

Consists of a symbol to be used to generate the error message and so that it can be referenced in the assembler cross reference for the indicated module. **This symbol must be specified.**

### BUFFER=

Specifies the address of a buffer that contains information concerning the disastrous error. This buffer information is traced. If BUFFER= is omitted no disastrous error information is traced.

### JOB=

Specifies the address of either the JCT or the JQE of the job being processed at the time the error occurred. If JOB= is specified, the job ID and job name are added to the start of the \$HASPO96 message.

Specify JOB= if you want the \$DISTERR service to determine what CB is present. Otherwise, specify JCT= or JQE=. Only one of JOB=, JQE=, or JCT= can be specified.

### JQE=

Functions the same as JOB=. However, the address of a JQE or JQA is passed. Only one of JOB=, JQE=, or JCT= can be specified.

### JCT=

Functions the same as JOB=. However, the address of a JCT is passed. Only one of JOB=, JQE=, or JCT= can be specified.

### DUMP=

Specifies whether a DUMP is desired. If NO, take no DUMP. If YES then use normal rules (RECVOPTs etc.) to determine if a dump should be taken. DUMP=YES is the default.

### MTTR

Specifies the address of the module track track record (MTTR) or a register that contains the address. The MTTR is recorded in a symptom record if a disastrous error occurs. MTTR= is mutually exclusive with MQTR=.

### MQTR

Specifies the address of the module quad track record (MQTR) or a register that contains the address. The MQTR is recorded in a symptom record if a disastrous error occurs. MQTR= is mutually exclusive with MTTR=.

## **\$DOGBERT**

### **RECVOPTS=**

Specifies the RECVOPTS option to use for this \$ERROR (JES2 main task). If not specified, the default is:

- DISTERR if DUMP=YES
- NODUMP if DUMP=NO

### **SIGRCD=**

Specifies the memory address of the signature record to be used in the SYMREC created as part of disastrous error processing. The record is normally the record for the track group in which MTTR (see above) appears, but it is up to the caller to specify what is desired.

This signature record is also used to determine whether its JQE is the same as the JQE passed as part of JOB= or JQE= keyword. If not the same, additional formatting is performed in the SYMREC creation.

## **Environment**

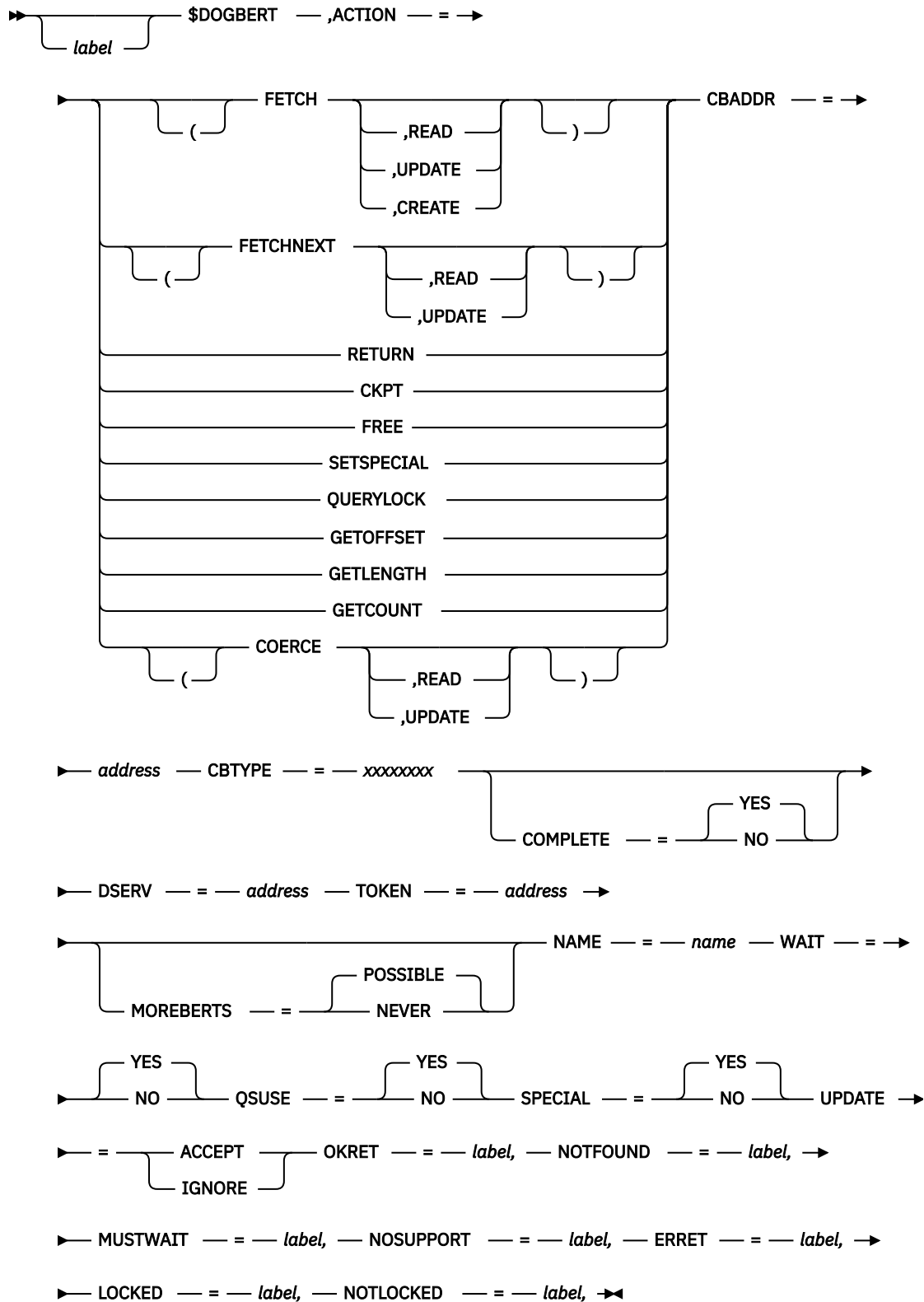
- Main task.
- \$WAIT can occur.

## **\$DOGBERT – Deliver or get BERT data**

---

Use the \$DOGBERT macro to copy data from the block extension reuse table (BERT) portion of the checkpoint into local storage or to return that data to the checkpoint.

## Format description



### ACTION=

Specifies the action to be taken. ACTION= is a required keyword. Valid values are:

**(FETCH,READ | UPDATE | CREATE)**

Requests that the BERTs described by the TOKEN= or NAME= be copied into the area pointed to by CBADDR. READ gets a read-only copy of the data. UPDATE gets a copy that can be updated and obtains the BERT lock. If READ or UPDATE is specified, then READ is assumed.

UPDATE is not allowed if DSERV= is specified. CREATE is the same as UPDATE but creates the named control block if it does not already exist. NAME= is required if CREATE is specified. READ is assumed if none of these actions (READ, UPDATE, or CREATE) is specified. UPDATE and CREATE are not allowed if DSERV= is specified.

Valid syntax is:

- ACTION=FETCH
- ACTION=(FETCH,READ)
- ACTION=(FETCH,UPDATE)
- ACTION=(FETCH,CREATE)

**(FETCHNEXT,READ | UPDATE)**

This action will do an implicit RETURN of the current CB into the BERT and then FETCH the next BERT in the chain. The READ and UPDATE option apply to the FETCH part of the operation. For the first control block, pass no token and ensure that the token in the prefix is zero. NAME= is not allowed for FETCHNEXT.

**CKPT**

This action is the same as RETURN except the BERT lock is **not** released. The changes that are made to the control block are scheduled to be written to the JES CKPT data set. There is no guarantee that the BERT has been written on this call.

**RETURN**

Requests that the CB passed be broken down to its BERTs and placed in the JES2 CKPT. The CB had to be obtained for UPDATE access. The BERT lock is released by this function. The TOKEN value returned from the FETCH (or FETCHNEXT) call must be passed on the RETURN call. An error return is taken (RC=16) if a control block fetched for READ access is passed to RETURN.

To create a new set of BERTs, do a RETURN with TOKEN= pointing to a ZEROed token value. BERTs are created and the token area updated.

**FREE**

The BERTs associated with the passed TOKEN= or NAME= (if supported) are freed. If the BERTs are chained, the specified BERT is dechained before being freed. If CBTOKEN= is passed, the CB must be in update mode.

**SETSPECIAL**

The SPECIAL attribute of an update mode control block is changed. Changing the SPECIAL attribute involves either getting or releasing the BERT lock. This request only updates the lock. No other data is updated.

**(COERCE,READ | UPDATE)**

The mode of the control block that is passed is changed to the mode specified without any update to the control block or the checkpointed BERTs.

**QUERYLOCK**

This action tests to see if the lock is available for the BERT whose token is passed as TOKEN=. If the lock is held, R15 is set to the member that owns the lock; R1 is the address of the PCE and R14 the address of the PREBERT if the owning member is our member. Otherwise, they are zero. The following exit labels are honored:

**LOCKED=**

Label branched to if the lock is held.

**NOTLOCKED=**

Label branched to if the lock can be obtained.



**ERRET=**

If specified, additional verification on the BERT type is done. This is label branched to if the validation fails.

**MUSTWAIT=**

If specified, additional checking is done to ensure there are free BERTs if one is needed for the lock. This is the label branched to if the lock is free, there are no BERTs for the CB, and there are no free BERTs (valid main task only).

**GETOFFSET**

Obtains the offset of a particular set of fields defined on a \$BERTTAB. CBTYPE= and NAME= are required and must match the CBTYPE= and NAME= parameters on the \$BERTTAB.

**GETLENGTH**

Obtains the length of a control block. The length is the highest offset used by any \$BERTTAB. CBTYPE= is required.

**GETCOUNT**

Obtain the count of how many instances of a keyed control block currently exist. CBTYPE= is required. The count is returned in register 1.

**CBADDR=**

Specifies the address of the control block storage. The control block must be prefixed with a \$PREBERT. This service validates and sets the appropriate fields in the prefix. If this is a FETCH or FETCHNEXT request, the BERTs are used to set fields in this area. If this is a CKPT or RETURN request, this area contains the data that is used to build the BERTs. If this is a FREE request, the prefix is validated before freeing the element. CBADDR= is required for ACTION=FETCH, FETCHNEXT, CKPT, and RETURN requests. CBADDR= is optional for FREE requests.

**CBTYPE=**

Specifies the type of control block to be processed. The type corresponds to a CBTYPE= on a \$BERTTAB macro. CBTYPE is **always** required. Valid values are:

- JQE
- CAT
- WSCQ
- INTERNAL

**Note:** CBTYPE=INTERNAL is used by the various BERT services and does not have a corresponding \$BERTTAB. The first two bytes of an internal control block must contain the total control block length.

**COMPLETE=**

Specifies how to handle the flag settings in the PREBERT. COMPLETE= is not valid if ACTION=QUERYLOCK, ACTION=GETOFFSET, or ACTION=GETLENGTH is specified. YES is the default.

**YES**

Completely creates the flag settings in the PREBERT. The flags are placed in PBEDGBF1 and PBEDGBF2.

**NO**

Updates the flag settings in the PREBERT set by the caller.

**DSERV=**

Specifies the address of the DSERV control block for the checkpoint version to be used. If not specified, the real CKPT is used. DSERV= is required if this is not the main task and not allowed if it is the main task. DSERV= is only valid for ACTION=FETCH or FETCHNEXT with read access being requested.

**Note:** You must obtain the IAZDSERV data area, which describes the checkpoint version, prior to invoking the \$DOGBERT service. Use a \$DSERV GET call to do so. See [Appendix D, “Accessing checkpoint control blocks outside the JES2 main task,”](#) on page 481 for a typical coding example.

**TOKEN=**

Specifies the 4-byte token that is associated with the first BERT in a set of BERTs. This is used as input for all services and as output for the FETCH, FETCHNEXT, CKPT, and RETURN services. For ACTION=

FETCH and FREE, either TOKEN= or NAME= must be specified. For ACTION= FETCHNEXT, CKPT, and RETURN (if TOKEN= is not specified), the current token from the \$PREBERT area is used. If specified as a register, then the register must point to a 4-byte token field.

If the value for TOKEN= is not valid (does not indicate the correct start of a control block) and NAME= is not specified, then depending on the DEBUG BERT= statement option, the service either ABENDs or returns a padded but otherwise null control block.

If TOKEN= and NAME= are both specified on a FETCH request, then the token is verified and used to locate the required BERTs. Then the NAME= is compared to the name in the control block located via TOKEN=. If the two values match, the control block is returned to the caller, which eliminates the need to search for control blocks by name.

If the token value is not valid, it is ignored, and the control block is located by name. If the control block is not found, an entry not found RC (4) is returned. If the control block located via TOKEN= does not match the specified NAME=, the TOKEN= value is ignored and the control block is located by the NAME= value.

**MOREBERTS=**

Specifies the action to take for BERTs. MOREBERTS= is not valid if ACTION=RETURN or ACTION=FETCHNEXT is specified. POSSIBLE is the default.

**POSSIBLE**

Indicates that the maximum number of BERTs that might be needed are obtained before any data is returned.

**NEVER**

Indicates that sufficient BERTs are already assigned for the block of data to return.

**NAME=**

Specifies the name of the BERT to be FETCHed or FREEd. The length of the name field is defined by the \$BERTTAB for the control block type. The value is used as input if specified on ACTION= FETCH or FREE calls. For ACTION=FETCHNEXT, NAME= specifies an optional output field to place the NAME in. NAME= is not allowed on an ACTION=CKPT request. For ACTION=OFFSET, the name is a specific name that should match the name coded on a \$BERTTAB.

For ACTION=FETCH, if NAME= and TOKEN= are both specified, then NAME= is used to validate the control block located by TOKEN=. Refer to the TOKEN= description for details.

For ACTION=FETCH, specifying NAME= only results in slower BERT retrieval times than if TOKEN= is specified. If large numbers of ACTION=FETCH requests are made with only NAME= when there are many instances of the same CBTYPE, performance can be negatively impacted. It is recommended to specify TOKEN= whenever possible minimize performance impact.

**WAIT=**

Specifies whether (YES) or not (NO) the \$DOGBERT service is allowed to \$WAIT. This parameter is only valid when DSERV= is not specified. If a \$WAIT is needed (for example, when not owning the queues or no BERTs available), the \$DOGBERT service returns to the caller (through the MUSTWAIT= label). The default is WAIT=YES.

**QSUSE=**

Specifies whether (YES) or not (NO) to obtain the CKPT queue lock before returning data. This is only valid if DSERV= is not specified and for FETCH and FETCHNEXT calls requesting read access to data. The default is QSUSE=YES.

**SPECIAL=**

Specifies whether (YES) or not (NO) special write processing is to be performed. Special processing grants write access to a control block but does not lock the element. The lock is checked and a wait is done if it is not available (and WAIT=YES is specified). On ACTION=RETURN, processing the lock is not released. A caller requesting SPECIAL=YES on a FETCH must also code SPECIAL=YES on the ACTION=RETURN. Also, the caller cannot \$WAIT between the FETCH and the corresponding RETURN. If the caller decides not to update the control block, the RETURN call is not required. The default is SPECIAL=NO. SPECIAL= is only valid on ACTION=FETCH or FETCHNEXT with UPDATE access and ACTION=RETURN.

# **UPDATE=ACCEPT | IGNORE**

For ACTION=RETURN and FETCHNEXT request (with previous BERT being obtained for UPDATE access), this keyword determines if any updates should be honored (ACCEPT) or ignored (IGNORE). If bypass is specified, the service only releases the lock associated with the BERT.

# **OKRET=**

Specifies the normal return branch label.

# **NOTFOUND=**

Specifies the label to branch to if a BERT that matches NAME= cannot be found. Also used when ACTION=FETCHNEXT and there are no more entries left.

# **MUSTWAIT=**

Specifies the label to branch to if the ACTION requires a \$WAIT and WAIT=NO was specified. For example, an ACTION=RETURN or CKPT that needs a new BERT when there are none free.

# **NOSUPPORT=**

Specifies the label to branch to if the current environment does not support BERTs. Checks of the prefix area are made prior to checking for BERT support.

# **ERRET=**

Specifies the default error branch label. This is used for return codes that do not have specific return code parameters, and in cases where NOTFOUND or MUSTWAIT are not specified.

# **ACTION=**

Table 4. \$DOGBERT Parameter Table (1 of 2)

	<b>FETCH, READ</b>	<b>FETCH, UPDATE</b>	<b>FETCHNEXT, READ</b>	<b>FETCHNEXT, UPDATE</b>	<b>CKPT</b>
CBADDR	Ri	Ri	Ri	Ri	Ri
CBTYPE	Ri	Ri	Ri	Ri	Ri
TOKEN	Oio	Oio	Oio	Oio	Oio
NAME*	Oi	Oi	Oo	Oo	X
DSERV	Oi	X	Oi	X	X
WAIT	Oi	X	Oi	Oi	Oi
QSUSE	Oi	X	Oi	X	X
SPECIAL	X	Oi	X	Oi	X
ERRET	X	X	X	Oi	X
NOTFOUND	O	O	O	O	X
MUSTWAIT	O	O	O	O	O
NOSUPPORT	O	O	O	O	O
ERRET	O	O	O	O	O
LOCKED	X	X	X	X	X
NOTLOCKED	X	X	X	X	X

Table 5. \$DOGBERT Parameter Table (2 of 2)

	<b>RETURN</b>	<b>FREE</b>	<b>SETSPECIAL</b>	<b>COERCE</b>	<b>QUERYLOCK</b>
CBADDR	Ri	Oi	Ri	Ri	X
CBTYPE	Ri	Ri	Ri	Ri	Ri

<i>Table 5. \$DOGBERT Parameter Table (2 of 2) (continued)</i>					
	<b>RETURN</b>	<b>FREE</b>	<b>SETSPECIAL</b>	<b>COERCE</b>	<b>QUERYLOCK</b>
TOKEN	Oio	Oio	Oio	Oio	Ri
NAME*	X	Oi	X	X	X
DSERV	X	X	X	X	Oi
WAIT	Oi	Oi	Oi	Oi	X
QSUSE	X	X	X	X	X
SPECIAL	Oi	X	Ri, Ni	Oi	X
ERRET	Oi	X	X	X	X
NOTFOUND	X	O	X	X	X
MUSTWAIT	O	O	O	O	X
NOSUPPORT	O	O	O	O	X
ERRET	O	O	O	O	O
LOCKED	X	X	X	X	O
NOTLOCKED	X	X	X	X	O

Keys:

X = Not valid      i = Input  
 O = Optional      o = Output  
 R = Required      N = Not required if COMPLETE=NO

**Note:** \* NAME= is only valid if the CB supports named references (as specified in the BERTTAB).

## Environment

- JES2 main task.
- \$WAIT can occur.

## Return codes

Return codes (R15 on exit) not ACTION=QUERYLOCK:

### Return Code

#### Meaning

**0**

BERT has been processed.

**4**

Entry not found.

**8**

\$WAIT needed but user coded WAIT=NO.

**12**

BERTs not supported at this time.

**16**

Other error.

Return codes (R15 on exit) ACTION=QUERYLOCK:

### Return Code

#### Meaning

**0**

Lock is not held.

**Not 0**

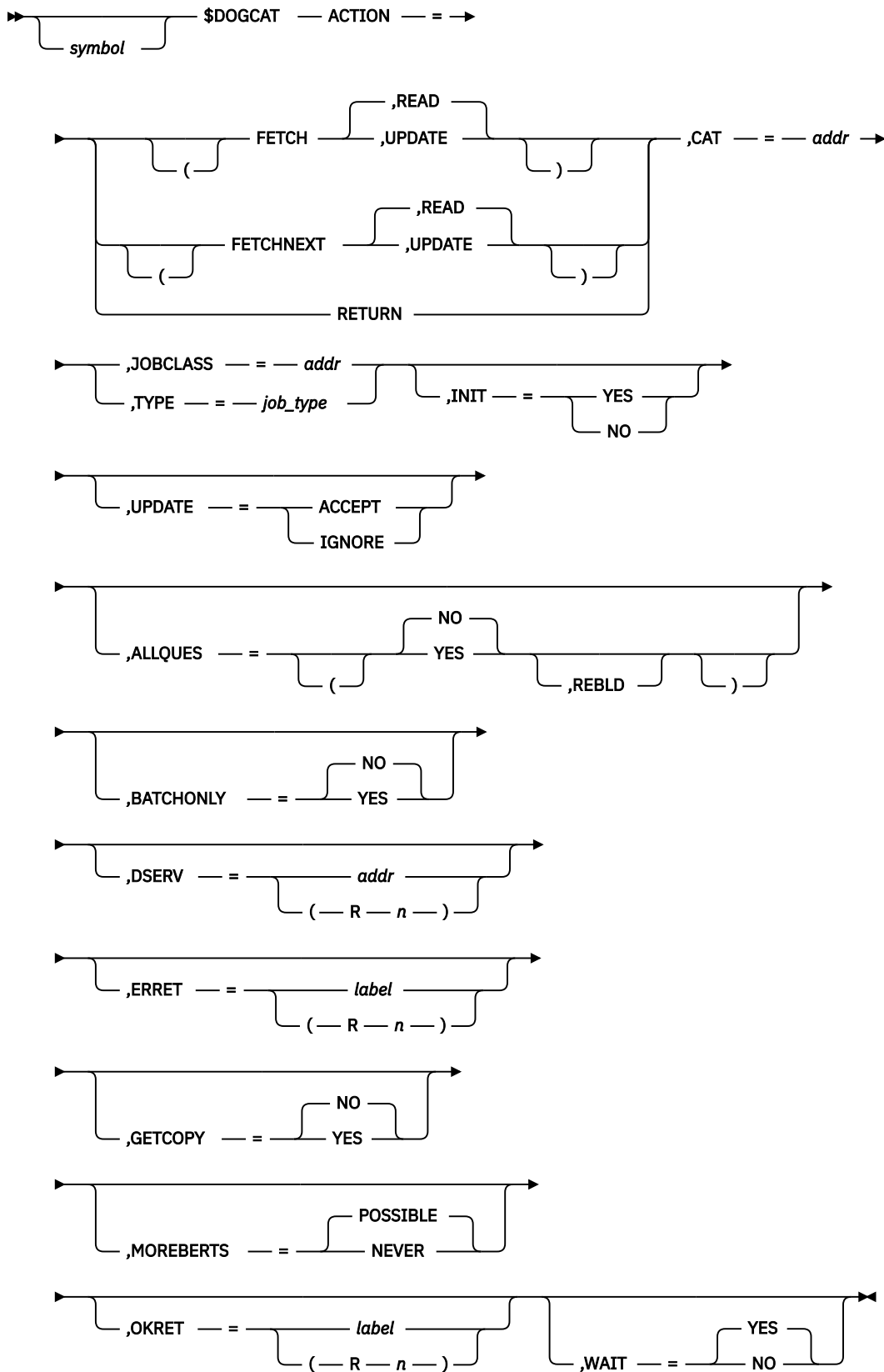
Member number that owns lock.

## **\$DOGCAT – Deliver or get CAT (class attribute table)**

---

Use \$DOGCAT to request that JES2 invoke class attribute table (CAT) processing to either return a copy of the CAT in a work area or to return the CAT to the checkpoint.

## Format description



### ACTION=

The function requested of the \$DOGCAT service.

**(FETCH,READ | UPDATE)**

Request that JES2 return the CAT for the specified job class.

**READ**

Indicates that you request a read-only copy of the CAT

**UPDATE**

Indicates that you request a copy of the CAT that you can update. JES2 locks the CAT using the block extension reuse table (BERT) lock until you return the updated copy.

**(FETCHNEXT,READ | UPDATE)**

Requests that JES2 get a copy current CAT and then "fetch" the next CAT in the chain.

**Note:**

1. JOBCLASS= is mutually exclusive with FETCHNEXT.
2. If you specify FETCHNEXT, CAT= is also required.
3. If the value of CAT= is zero, JES2 returns the first CAT in the chain.

**Note:** If ACTION=(FETCHNEXT,READ) and your logic is such that a \$WAIT can occur, you can ensure that the CAT has the most up to date information by specifying GETCOPY=YES.

**RETURN=**

Specifies that JES2 should unlock the CAT, the NEW values in the CAT written to the checkpoint, and the work space storage freed.

**JOBCLASS=**

Specifies the address of an 8-byte field that contains the job class for which the CAT is to be obtained. The job class should be left-aligned and padded with blanks.

JOBCLASS= is mutually exclusive with TYPE=.

**TYPE=**

Specified the job type for which the CAT is to be obtained.

TYPE= is mutually exclusive with JOBCLASS=.

**INIT= YES | NO**

Specifies whether (YES) or not (NO) to build a CAT using the initialization (that is, local copy) of the CAT control.

**UPDATE=ACCEPT | IGNORE**

Specifies for ACTION=RETURN and FETCHNEXT request (with previous CAT having been obtained for UPDATE access) whether JES2 should honor (ACCEPT) or ignore (IGNORE) an updates to the CAT.

**DSERV=**

Specifies the address of the DSERV control block for the checkpoint version you request JES2 to use. If you do not specify DSERV=, JES2 uses the real CKPT.

**Note:**

1. DSERV= is not allowed in the JES2 main task environment.
2. You must specify DSERV= in User, Subtask, and FSS environments.
3. You must obtain the IAZDSERV data area, which describes the checkpoint version, prior to invoking the \$DOGCAT service. Use a \$DSERV GET call to do so. See [Appendix D, "Accessing checkpoint control blocks outside the JES2 main task," on page 481](#) for a typical coding example.

**(ALLQUES=YES | NO,REBLD)**

Specifies whether (YES) or not (NO) JES2 returns all job queues. If you specify ALLQUES=YES, JES2 returns a CAT will be returned for each of the queue heads in the JQE.

ALLQUES=YES is only allowed when you also specify ACTION=FETCHNEXT.

When ALLQUES=YES is specified, you can also specify a second positional parameter, REBLD, to indicate that JES2 also returns the JQE rebuild queue as a queue head. You cannot specify ALLQUES with BATCHONLY=YES.

**BATCHONLY=YES | NO**

For ACTION=FETCHNEXT, returns only batch execution class CATs (BATCHONLY=YES) or returns all execution queues (BATCHONLY=NO). The default value is BATCHONLY=NO. BATCHONLY is only valid for ACTION=FETCHNEXT. You cannot specify BATCHONLY with ALLQUES=YES.

**CAT=**

For ACTION=FETCH, JES2 obtains and returns the address of the CAT to the caller. The caller should not provide a work area on this type of call.

For ACTION=RETURN, the caller provides the address of the CAT that JES2 will return.

**ERRET=**

Specifies a label to be branched to or a register to be branched on if a non-zero return code is returned in R15.

**GETCOPY=YES | NO**

For ACTION=(FETCH,READ), returns a copy of the cached CAT (GETCOPY=YES) or returns the cache entry (GETCOPY=NO). To limit resource usage, JES2 typically returns a CAT that is in a cache, even for read access. It is expected that the caller will not update this read mode CAT. However, if the caller needs to make impermanent updates to the read only CAT, a copy of the CAT must be obtained by specifying GETCOPY=YES. Failure to get a copy when updating a read mode CAT could result in an inconsistent CAT structure and various errors.

If GETCOPY=YES is provided on the first of a series of (FETCHNEXT, READ), then GETCOPY is in effect for the entire series. If the cache should become stale while the series is in progress, the cache is refreshed and the next class in the cache, after the prior one in the series, is returned. If the prior class is no longer in the cache, the class that would have been next in the collating sequence is returned.

**MOREBERTS=**

Specifies the action to take for BERTs. MOREBERTS= is not valid if ACTION=RETURN or ACTION=FETCHNEXT is specified. POSSIBLE is the default.

**POSSIBLE**

Indicates that the maximum number of BERTs that might be needed are obtained before the CAT is returned.

**NEVER**

Indicates that sufficient BERTs are already assigned for the CAT to return.

**OKRET=**

Specifies a label to be branched to or a register to be branched on if a zero return code is returned in R15.

**WAIT=**

Specifies whether (YES) or not (NO) the caller can \$WAIT when the CAT is returned. WAIT= can be specified for ACTION=(FETCH,UPDATE) or ACTION=(FETHNEXT,UPDATE). WAIT= is not valid if DSERV= is specified. YES is the default.

**YES**

Indicates that the caller can \$WAIT when the CAT is returned.

**NO**

Indicates that the caller cannot \$WAIT when the CAT is returned. If there are not enough BERTs for a maximum sized CAT to return, no CAT is obtained and the return code is set to 8.

## Environment

- Main Task - all actions are available
- USER, SUBTASK and FSS environment - only FETCH and FETCHNEXT for READ access

## Return codes

The following return codes (in decimal) are returned in register 15.



**Return Code  
Meaning**

- 0** CAT processed successfully
- 4** CAT not found
- 8** CAT not returned because of insufficient BERTs

On return, register 1 contains the address of the CAT returned.

## **\$DOGDJB – Deliver or Get DJB**

---

This macro invokes the DJB (duplicate job block) processing services to perform one of the following functions:

**FETCH**

Returns a copy of the DJB in a work area.

**FETCHNEXT**

Returns a copy of the next DJB in a work area.

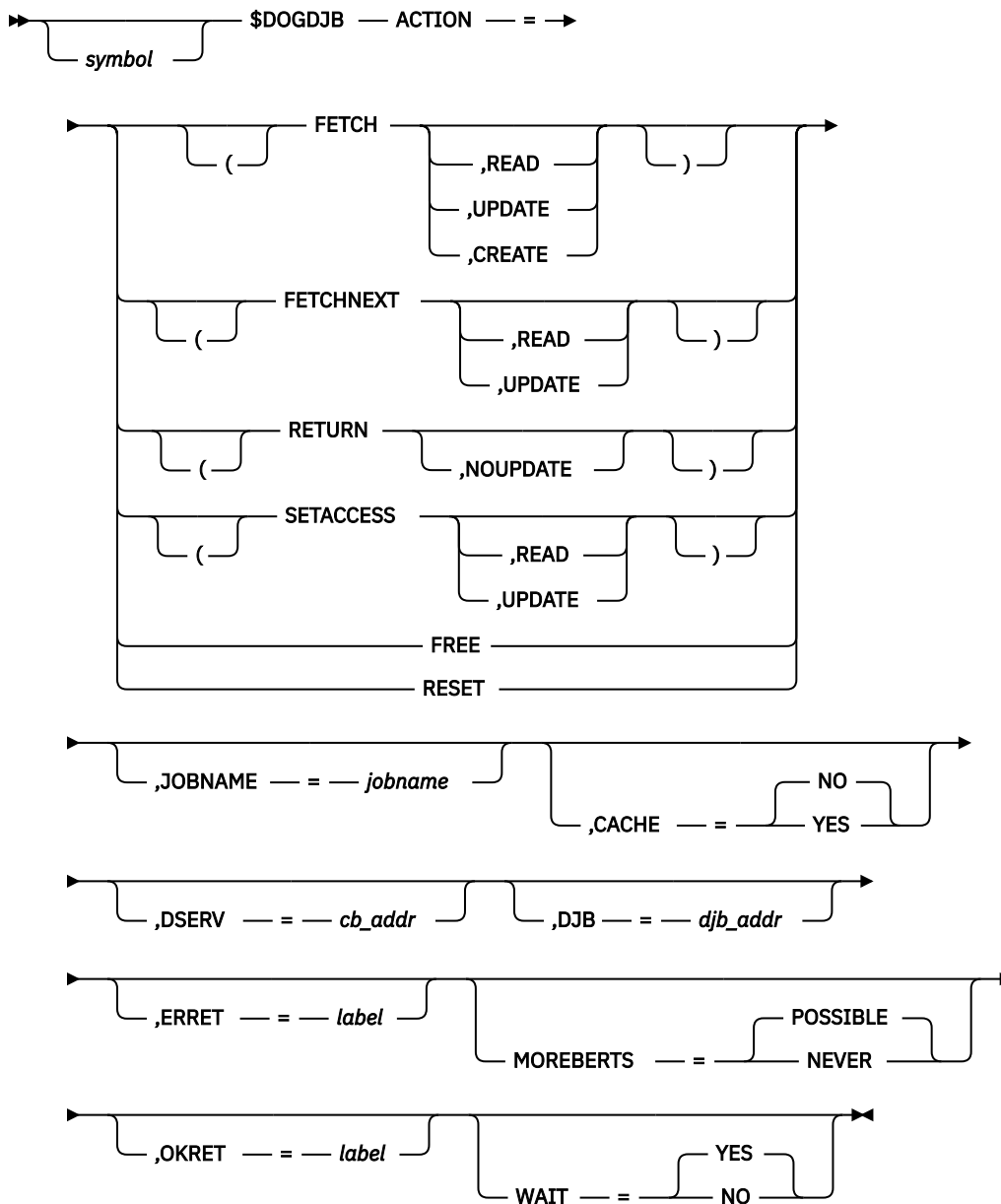
**RETURN**

Returns the DJB to the checkpoint.

**FREE**

Removes the DJB from the checkpoint.

## Format description



### ACTION=

The action to be taken. ACTION= is a required keyword. The following values are the valid values for ACTION=:

#### (FETCH,READ|UPDATE|CREATE)

Request that the DJB for the specified jobname be returned. READ gets a read-only copy of the DJB. UPDATE gets a copy that can be updated and locks it through the BERT lock. CREATE gets an UPDATE DJB asking that one be created if none exists yet. If none of READ, UPDATE, or CREATE is specified, READ is assumed. UPDATE is not allowed if DSERV= is specified.

#### (FETCHNEXT,READ|UPDATE)

This action performs an implicit DELIVER of the current CB into the BERT and then FETCH the next BERT in the chain. The READ and UPDATE options apply to the FETCH part of the operation.

#### RETURN

Requests that the DJB be unlocked, the NEW values in the DJB written to the checkpoint, and the storage freed. NOUPDATE requests that the new values are not to be written.

**(SETACCESS, UPDATE|READ)**

Requests that the DJB mode be changed to UPDATE or READ. This requires that a DJB address is specified on DJB=.

**FREE**

The BERTs associated with this DJB are freed.

**RESET**

Requests that the READ mode DJB cache be freed.

**JOBNAME=**

The jobname for which the DJB is to be obtained.

**CACHE=**

Specifies whether (YES) or not (NO) to manage the DJB by using the read-mode cache. CACHE is valid only when ACTION=FETCH is specified. NO is the default.

**YES**

Manages this DJB by using the read-mode cache. If a DJB address is given when ACTION=(FETCH), the DJB address that is returned in R1 might differ from the given DJB address. If a \$WAIT occurs while a READ mode DJB is active, the DJB can be returned during the \$WAIT.

**NO**

Do not use the read-mode cache to manage the DJB.

**DSERV=**

Address of the DSERV control block for the checkpoint version to be used. If DSERV= is not specified, the real CKPT is used. DSERV= is required if this is not the main task and not allowed if it is the main task. DSERV= is only valid for ACTION=FETCH or ACTION=(FETCH,READ).

**DJB=**

Address of DJB to return (RETURN), or storage into which the DJB should be placed (FETCH), or the DJB whose access mode is changed (SETACCESS).

**ERRET=**

Default error branch label.

**MOREBERTS=**

Specifies the action to take for BERTs. MOREBERTS= is not valid if ACTION=RETURN or ACTION=FETCHNEXT is specified. POSSIBLE is the default.

**POSSIBLE**

Indicates that the maximum number of BERTs that might be needed are obtained before the DJB is returned.

**NEVER**

Indicates that sufficient BERTs are already assigned for the DJB to return.

**OKRET=**

Normal return branch label.

**WAIT=**

Specifies whether (YES) or not (NO) the caller can \$WAIT when the DJB is returned. WAIT= can be specified for ACTION=(FETCH,UPDATE), ACTION=(FETCH,CREATE), or ACTION=(FETHNEXT,UPDATE). WAIT= is not valid if DSERV= is specified. YES is the default.

**YES**

Indicates that the caller can \$WAIT when the DJB is returned.

**NO**

Indicates that the caller cannot \$WAIT when the DJB is returned. If there are not enough BERTs for a maximum sized DJB to return, no DJB is obtained and the return code is set to 8.

## Environment

All actions are available in the JES2 Main task. FETCH for READ access is available in USER, SUBTASK and FSS environment.

## Registers on entry

**R0 - R10:**

N/A

**R11:**

HCT or HCCT, depending on JES2 environment (JES2 main task or USER environment).

**R12:**

N/A

**R13**

PCE address or usable save area, depending on JES2 environment (JES2 Main task or USER environment).

**R14 - R15:**

N/A

## Registers on exit

**R0:**

Unchanged

**R1:**

DJB address

**R2 - R13:**

Unchanged

**R14**

Destroyed

**R15:**

Return code

## Return codes

The following return codes (in decimal) are returned in register 15.

**Return Code****Meaning****0**

DJB processed successfully

**4**

DJB not found

**8**

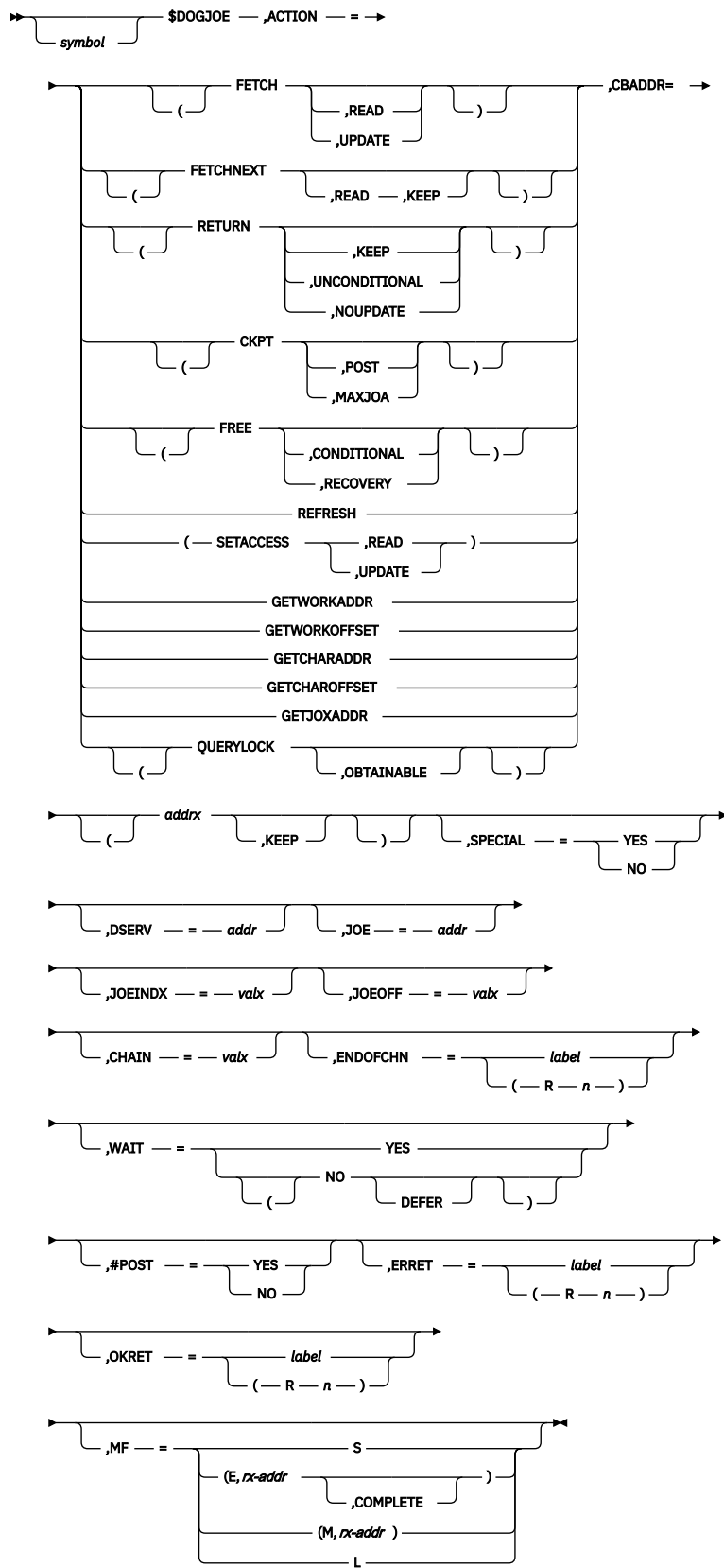
DJB not returned because of insufficient BERTs

## **\$DOGJOE – Deliver or get JOE**

---

Use \$DOGJOE to request that JES2 builds or returns an artificial JOE. An artificial JOE consists of the base JOE, the characteristics JOE, and the additional fields defined in the JOA (BERT and JOX backed data areas).

## Format description



### ACTION=

The action that you request JES2 to take. Valid values are:

**(FETCH,mode)**

Requests that JES2 constructs an artificial JOE (JOA) in the area that CBADDR points to. The optional second operand *mode* can be one of the following values:

**READ**

Gets a read-only copy of the data. This is the default.

**UPDATE**

Gets a copy of the data that you can update. UPDATE is not valid if DSERV= is specified.

**Note:** On return from a (FETCH,UPDATE) request, the queues (\$QSUSE call) are owned.

If a ACTION=FETCH call is made, the caller can provide a real JOE, a read mode JOA, or an update mode JOA. If a real JOE or a read mode JOA is provided, a new JOA is obtained for the caller. If an update mode JOA is provided, that same JOA is passed back to the caller. The service also tracks the callers (within the invocation stack) who are using the same JOA.

**Note:** You cannot specify WAIT=YES if ACTION=(FETCH,READ). If ACTION=(FETCH,READ), the queues are not necessarily owned unless the caller does a \$QSUSE.

**(FETCHNEXT,READ,KEEP)**

Requests that the next JOE on the chain, which is indicated by the CHAIN= keyword, refreshes the artificial JOE in the area that CBADDR points to. The optional READ operand indicates that the JOE is in READ mode.

If there are no remaining JOEs, the call is converted to ACTION=RETURN. If the KEEP operand is also specified, the memory for the JOA will not be released. Control is passed to the point specified by the ENDOFCHN= value.

**Note:** You cannot specify WAIT=YES if ACTION=(FETCHNEXT,READ). If ACTION=(FETCHNEXT,READ), the queues are not necessarily owned unless the caller does a \$QSUSE.

**RETURN**

Requests that JES2 releases the storage for the JOA. If the JOA was obtained with UPDATE access, the JOA will be broken down and all of its component parts will be updated in the checkpoint.

Optionally, a second operand can be supplied with RETURN:

**(RETURN,KEEP)**

Indicates that JES2 will not release the memory for the JOA.

**(RETURN,UNCONDITIONAL)**

Indicates that JES2 releases the BERT lock and memory for the JOA even if there are users of the JOA in the USER stack. JES2 also releases the USER stack.

**(RETURN,NOUPDATE)**

Indicates that JES2 pops the current stack element without checkpointing the JOE. If there are no stack elements, this keyword is equivalent to FREE.

**CKPT**

Requests that JES2 breaks down the JOA and places the component parts into the JES2 checkpoint (CKPT). JES2 should obtain the JOA for UPDATE access, but does not release it.

Optionally, a second operand can be supplied with CKPT:

**(CKPT,POST)**

Indicates that a \$POST CKPTW is executed after the data is placed in the CKPT.

**(CKPT,MAXJOA)**

Indicates that the \$DOGBERT macro should ensure that the maximum number of BERTs, which might be needed when the JOA is returned, is available.

**Note:** If the passed CBADDR is not a JOA, it is assumed a JOE. The real JOE will be checkpointed (\$CKPT call).

**FREE**

Requests that JES2 releases the artificial JOE passed on CBADDR= without updating the checkpoint. With this keyword, JES2 unlocks the BERTs, releases the JOA memory, and releases all the user elements.

Optionally, a second operand can be supplied with FREE:

**(FREE,CONDITIONAL)**

Performs the FREE operation only if the JOA is not already free.

**(FREE,RECOVERY)**

Indicates that JES2 finds and releases the JOA that is created by calling PCE for the real JOE. This keyword finds the JOA whose address is lost for recovery routines. The input is provided by the JOE= keyword instead of the CBADDR= keyword.

**REFRESH**

Requests that JES2 refreshes the artificial JOE from the checkpoint. This keyword does nothing if a real JOE or an update mode JOA is passed in.

**(SETACCESS,mode)**

Requests that JES2 sets the access mode of the JOA to *mode*. If the JOA is already in the requested access mode, the request is ignored.

**Note:** A SETACCESS request might free the JOA passed in and obtain a new JOA. The caller should always refresh the JOA address with the value passed back in register 0 (R0).

The second operand *mode* can be one of the following values:

**READ**

Indicates that JES2 sets the JOA to READ only mode. The data that already exists in the JOA is copied to the checkpoint.

**Note:** A Jxx abend occurs if SETACCESS,READ is specified against a JOA with a stack of users.

**UPDATE**

Indicates that JES2 sets the JOA to UPDATE mode. The JOA is refreshed from the checkpoint data.

**GETWORKADDR**

Requests that JES2 computes the address of the WORK JOE represented by the passed work JOE or JOA. This keyword is valid only if MF=S (S is the default).

**Note:** If the caller gives an invalid block of memory as a JOA, the results are unpredictable.

**GETWORKOFFSET**

Requests that JES2 computes the offset of the WORK JOE represented by the passed work JOE or JOA. This keyword is valid only if MF=S (S is the default).

**Note:** If the caller gives an invalid block of memory as a JOA, the results are unpredictable.

**GETCHARADDR**

Requests that JES2 computes the address of the characteristics JOE represented by the passed work JOE or JOA. This keyword is valid only if MF=S (S is the default).

**Note:** If the caller gives an invalid block of memory as a JOA, the results are unpredictable.

**GETCHAROFFSET**

Requests that JES2 computes the offset of the characteristics JOE represented by the passed work JOE or JOA. This keyword is valid only if MF=S (S is the default).

**Note:** If the caller gives an invalid block of memory as a JOA, the results are unpredictable.

**GETJOXADDR**

Requests that JES2 computes the address of the JOE extension represented by the passed work JOE or JOA. This keyword is valid only if MF=S (S is the default).

**Note:** If the caller gives an invalid block of memory as a JOA, the results are unpredictable.

**QUERYLOCK**

Queries whether JES2 holds the BERT lock for the JOE. An RC of 0 means NO. A positive RC means YES and the value is the member number where the lock is held. A second optional operand can be supplied with QUERYLOCK:

**(QUERYLOCK,OBTAINABLE)**

Queries whether JES2 holds the BERT lock for the JOE and if the lock is not held, tests whether there are any BERTs available. If there are no BERTs, the lock cannot be obtained and an RC of -1 is returned.

**Note:** A CBADDR= keyword or one of the JOE=, JOEINDX=, and JOEOFF= keywords is required.

**CBADDR=(addrx,KEEP)**

Specifies the address of an artificial JOE. This area is allocated by using the \$GETWORK macro. Normally, it is allocated by the service if CBADDR is not specified with ACTION=FETCH. If CBADDR is specified with ACTION=FETCH, caller-provided area is used. This parameter must be specified for most ACTIONS, see description of individual ACTIONS for exceptions. A second operand of KEEP can be specified if ACTION=RETURN:

**KEEP**

Retains the memory for CBADDR.

**Note:**

1. If ACTION=FETCH, JES2 uses the data area that is specified by CBADDR=, but not a new area assigned by the \$GETWORK macro.
2. This keyword is not valid if MF=L or MF=M.

**SPECIAL=YES|NO**

Specifies whether a JOA is built with (NO) or without (YES) obtaining the BERT lock. If SPECIAL=YES is specified, no \$WAITs will occur.

**Note:** SPECIAL=YES is valid only if ACTION=(FETCH,UPDATE).

**DSERV=addr**

Specifies the address of the DSERV control block that the checkpoint version uses. If DSERV= is not specified, the real CKPT is used.

**Note:**

1. DSERV= is required if this is not the main task and is not allowed if this is the main task.
2. DSERV= is only valid for ACTION=FETCH with read access requested.
3. DSERV= cannot be specified if MF=L or MF=M.

**JOE=**

Specifies the address of a work JOE or a JOA. JOE=, JOEINDX= or JOEOFF= is required for ACTION=FETCH, otherwise it is not valid. JOE= is mutually exclusive with JOEINDX= and JOEOFF=. If JOE= is the address of a work JOE or the address of a READ mode JOA, a new JOA is constructed. If JOE= is the address of a UPDATE mode JOA, a PBEUSER stack element is created to represent this request and the JOA is returned to the caller.

**Note:** JOE= cannot be specified if MF=L or MF=M.

**JOEINDX=**

Specifies the index of the work JOE. JOE=, JOEINDX= or JOEOFF= is required for ACTION=FETCH, otherwise it is not valid. JOEINDX= is mutually exclusive with JOE= and JOEOFF=.

**Note:** JOEINDX= cannot be specified if MF=L or MF=M.

**JOEOFF=**

Specifies the offset of work JOE. JOE=, JOEINDX= or JOEOFF= is required for ACTION=FETCH, otherwise it is not valid. JOEOFF= is mutually exclusive with JOE= and JOEINDX=.

**Note:** JOEOFF= cannot be specified if MF=L or MF=M.



**CHAIN=**

Specifies the offset of field which is used to find the next JOE. The field must be within the artificial JOE. CHAIN= is required only if ACTION=FETCHNEXT.

**ENDOFCHN=**

Specifies a label or a register that contains an address to which JES2 branches if an end-of-chain condition exists for ACTION=FETCHNEXT. If JES2 reaches the JOE end-of-chain, JES2 frees the JOA.

**Note:** This parameter cannot be specified if MF=L or MF=M.

**WAIT=YES|NO**

Specifies whether (YES) or not (NO) JES2 waits for any needed service or returns an error if a resource is not available. YES is the default unless ACTION=(FETCH,READ) or ACTION=(FETCHNEXT,READ). If a WAIT is necessary within the \$DOGBERT service and WAIT=NO is specified, the requested ACTION will not be done.

If WAIT=NO is specified, an optional second operand of DEFER can be specified. WAIT=(NO,DEFER) indicates that JES2 cannot tolerate a \$WAIT macro, but automatically defers the action if a \$WAIT service is necessary. The JOA is returned by the \$DILBERT service if necessary. WAIT=(NO,DEFER) is valid only if ACTION=RETURN.

**Note:** WAIT=YES is mutually exclusive with ACTION=QUERYLOCK.

**#POST=YES|COND**

Specifies the conditions under which the \$#POST routine will be called.

**YES**

A call to \$#POST is made regardless of other factors.

**COND**

\$#POST is only called if the owner of an update mode JOA has \$WAITed since the JOA was locked. This is the default.

A second operand of KEEPJWEL can be specified to indicate that the JWELs and TJEVs will not be deleted if a \$#POST is done.

**Note:**

1. #POST= is valid only if ACTION=RETURN or ACTION=CKPT.
2. \$#POST processing occurs when the final RETURN of an update mode JOA is made.

An additional operand of JOETIME can be specified to indicate that JOECRTME should be updated if the installation has specified OUTDEF OUTTIME=UPDATE.

**ERRET=**

Specifies a label to be branched to or a register to be branched on if a nonzero return code is returned in R15.

**Note:** This parameter cannot be specified if MF=L.

**OKRET=**

Specifies a label to be branched to or a register to be branched on if a zero return code is returned in R15.

**Note:** This parameter cannot be specified if MF=L.

**MF=**

Specifies the macro form.

**MF=S**

Standard (inline) form.

**MF=(E,rx-addr)**

Execute form. An optional third operand of COMPLETE can be specified.

**MF=(M,rx-addr)**

Modify form.

## \$DOGJQE

**MF=L**

List form.

### Return codes

The following return codes (in decimal) are returned in register 15.

Return Code	Meaning
-------------	---------

0	Processing successful (no errors).
---	------------------------------------

4	ACTION=FETCHNEXT processing encountered end of chain and KEEP JOA was specified, or ACTION=CKPT processing was successful even though CBADDR specifies the address of a real JOE.
---	---

8	WAIT=NO was specified and a WAIT was required. ACTION was not performed.
---	--

12	WAIT=YES was specified and JOE disappeared while waiting for BERT lock.
----	---

NZ	Member number holding the BERT lock if ACTION=QUERYLOCK was specified (0 if lock was not held).
----	---

-1	QUERYLOCK and lock were available, but the attempt to lock JOE failed because of a BERT shortage. Only returned if ACTION=(QUERYLOCK,OBTAINABLE).
----	---

### Environment

- Main task (all actions are available).
- USER, SUBTASK, and FSS environments (ACTION=FETCH for READ access only).
- \$WAIT can occur.

## \$DOGJQE – Deliver or get JQE

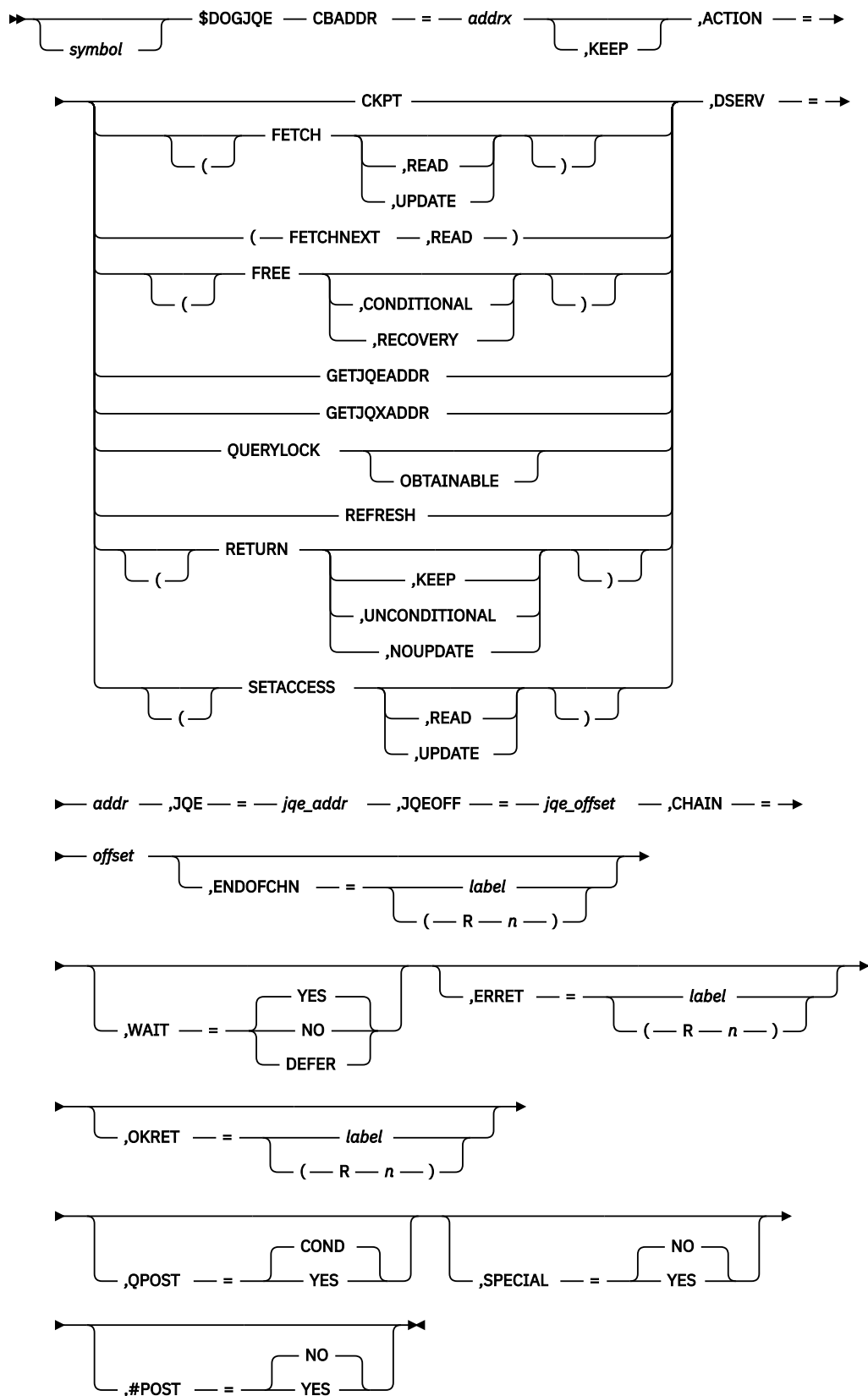
---

Use \$DOGJQE to request that JES2 build or return an artificial JQE. An artificial JQE consists of the base JQE, the JQX, and the additional fields defined in the JQA.

When a call is made with ACTION=FETCH, the caller can provide a real JQE, a read-mode JQA, or an update mode JQA. If a real JQE or read-mode JQA is provided, then a new JQA is obtained for the caller. The JQA returned from \$DOGJQE will always be a z2-mode JQA. Therefore, the caller of \$DOGJQE does not have to determine the mode of the checkpoint (R4 or z2) before accessing fields in the JQA.

If a FETCH call is made and the input is an update-mode JQA, then that same JQA is passed back to the caller. Transparently to the caller, a stack element is created to keep track of the callers who are using the same JQA. The elements are kept in a USER stack and are taken off the queue in a last in, first out manner when the caller makes a RETURN call. All stack elements are removed if the caller makes a FREE call or (RETURN,UNCONDITIONAL).

## Format description



**CBADDR=addrx[,KEEP]**

Address of an artificial JQE (JQA).

On an ACTION=FETCH call, JES2 obtains and returns this address to the caller. The caller should not provide a work area on this type of call unless it is a work area whose address was returned through R0 on some prior \$DOGJQE ACTION=FETCH call. JQAs obtained through ACTION=FETCH must eventually be released through ACTION=RETURN (without the KEEP operand) or ACTION=FREE.

On an ACTION=RETURN|CKPT|FREE call, the caller provides the address of the artificial JQE (JQA). On an ACTION=CKPT call JES2 assumes that the CBADDR you pass is a JQA or a JQE. If the CBADDR you pass is not a JQA for ACTION=RETURN or FREE, a Q21 abend will occur.

CBADDR is the address of a JQA and is a required parameter on ACTION=RETURN, CKPT, and FREE calls. If CBADDR is not provided on an ACTION=FETCH call, by default JES2 obtains the required storage and returns the address of the JQA in register 0. If CBADDR is provided on an ACTION=FETCH call and the JQA is in the wrong mode (it's a READ mode JQA and the current request is for an UPDATE mode or vice-versa), then the provided JQA is released and a new JQA is obtained in the proper mode.

**KEEP**

Requests that JES2 retain the memory for CBADDR. This is only valid for ACTION=RETURN calls. The retained memory can be passed in on the CBADDR parameter for an ACTION=FETCH call. JES2 uses the data area passed in rather than obtaining a new work area.

**ACTION=**

The action you request JES2 to take. Valid values are:

**CKPT**

Requests JES2 breaks down the artificial JQE into its component parts and places them into the JES2 checkpoint (CKPT). You must also obtain the JQA for UPDATE access; JES2 does not release it after use.

Optionally ACTION=(CKPT,POST) can be coded. If POST is provided, a \$POST CKPTW is executed after the data is placed in the CKPT.

**Note:** If the CBADDR passed is not a JQA, it is assumed to be a JQE and JES2 will checkpoint the real JQE.

**(FETCH)**

Requests that JES2 construct an artificial JQE. By default, JES2 obtains and returns the address of the artificial JQE to the caller. See CBADDR= for more information. FETCH always returns a JQA that is in z2 mode.

Optionally, a second operand can be supplied with FETCH:

**(FETCH,READ)**

Gets a read-only copy of the data.

**(FETCH,UPDATE)**

Gets a copy of the data that you can update. You cannot specify UPDATE if you also provide a job information block through the DSERV= keyword.

**Note:** A \$QSUSE must be in effect when this call is made and the access requested is UPDATE.

**(FETCHNEXT,READ)**

Request that JES2 refresh (using the next JQE found by using the CHAIN= field provided) the artificial JQE in the area pointed to by CBADDR=. If FETCHNEXT is specified, you must also specify CHAIN=. If there are no more JQEs on the chain, JES2 converts the call to an ACTION=RETURN call and passes control to the point specified by ENDOFCHN=.

**(FREE)**

Requests that the artificial JQE be released without updating the checkpoint. As a result, the BERTs are unlocked, the JQA memory is freed, and all the user stack elements are freed. The address of the JQA is passed by the CBADDR= keyword.

Optionally, a second operand can be supplied with FREE:

**(FREE,CONDITIONAL)**

Performs the FREE operation only if the JQA memory has not already been freed.

**(FREE,RECOVERY)**

Indicates that the JQA created by the calling PCE for the real JQE is found and freed. This operand is for recovery routines that need to find a JQA whose address is unknown. Input is given through the JQE= keyword, not the CBADDR= keyword.

**Note:** Only use ACTION=FREE as part of a \$ESTAE recovery routine. JES2 will issue a \$ERROR \$DJ1 if you request ACTION=FREE and IBM-supplied code is also a user of the JQA.

**GETJQEADDR**

Requests that JES2 determine the address of the JQE represented by the artificial JQE.

**GETJQXADDR**

Requests that the address of the real JQX represented by the artificial JQE be computed.

**QUERYLOCK**

Requests that JES2 determine if the BERT lock is held for the JQE.

If you specify QUERYLOCK, you must also specify either CBADDR=, JQE=, or JQEOFF.

**QUERYLOCK,OBTAINABLE**

Indicates that the caller needs to know if the BERT lock is free and there is a BERT available to obtain the lock. If the BERT lock is available, but there are no BERTs available in which to place the lock, JES2 returns a return code of -1 to the caller. In all other cases the return from this form of the QUERYLOCK is identical to ACTION=QUERYLOCK.

**REFRESH**

Requests that JES2 refresh the artificial JQE from the checkpoint

**(RETURN)**

Requests that the artificial JQE be broken down and the component parts be placed into the JES2 checkpoint (CKPT). The component parts of the artificial JQE will be placed into the checkpoint only if the artificial JQE is in UPDATE mode. If the JQA is obtained in READ mode, JES2 releases the memory of the artificial JQE.

Optionally, a second operand can be supplied with RETURN:

**(RETURN,KEEP)**

Requests that JES2 retain the memory for CBADDR. This retained memory can then be passed in on the CBADDR parameter for an ACTION=FETCH call. JES2 will then use the data area passed in rather than obtaining a new work area.

**(RETURN,UNCONDITIONAL)**

Specifies that the BERT lock and memory for the JQA will be released regardless of the number of apparent users of the JQA in the user stack. The user stack elements are released as well.

**(RETURN,NOUPDATE)**

Specifies that the JQA is to be returned but the JQA itself is not to be written to the checkpoint. If USER stack elements exist, a single stack element will be popped. If no user stack elements exist, this is equivalent to ACTION=FREE.

**Note:** Any changes made to the artificial JQE will NOT be backed out.

**(SETACCESS)**

Requests that JQA be placed into UPDATE mode, implying that the JQA will be refreshed from the checkpoint data.

**Note:** A SETACCESS request might free the passed-in JQA and obtain a new JQA. The caller should always refresh the JQA address with the value passed back in register 0 (R0).

Optionally, a second operand can be supplied with SETACCESS:

**(SETACCESS,READ)**

JES2 makes the JQA READ-only, implying that the already existing data in the JQA will be copied to the checkpoint.

**Note:** Requesting a SETACCESS,READ against a JQA with a stack of users results in a Q19 abend.

**(SETACCESS,UPDATE)**

JES2 places the JQA into UPDATE mode implying that the JQA will be refreshed from the checkpoint data.

**Note:** A \$QSUSE must be in effect when this call is made and the access requested is UPDATE.

**DSERV=**

Address of the JES job information service token list (DSERV) control block for the checkpoint version to be used. If not specified, JES2 uses the real CKPT.

**Note:**

1. DSERV= is required if this is not the main task environment.
2. DSERV= is not allowed if this is the main task environment.
3. DSERV= is only valid for ACTION=FETCH with read access requested.
4. You must obtain the IAZDSERV data area, which describes the checkpoint version, prior to invoking the \$DOGJQE service. Use a \$DSERV GET call to do so. See [Appendix D, "Accessing checkpoint control blocks outside the JES2 main task,"](#) on page 481 for a typical coding example.

**JQE=**

Address of the JQE or JQA. JQE= or JQEOFF= is required for ACTION=FETCH calls and optional for ACTION=QUERYLOCK, otherwise it is not valid. JQE= is mutually exclusive with JQEOFF=.

If JQE is the address of a JQE or the address of a READ mode JQA, then a new JQA is constructed. If JQE is the address of an UPDATE mode JQA, then that JQA is updated with an address of a user stack element.

**JQEOFF=**

Offset of the JQE. JQE= or JQEOFF= is required for ACTION=FETCH calls only. JQEOFF= is mutually exclusive with JQE=.

**CHAIN=**

Offset of field which is to be used to find the next JQE. The field must be within the artificial JQE. This specification is required for ACTION=FETCHNEXT calls only.

**ENDOFCHN=**

Specifies a label or a register that contains an address to which JES2 branches if an end-of-chain condition exists for ACTION=FETCHNEXT. If JES2 reaches the JQE end-of-chain, JES2 frees the JQA.

**WAIT=**

Specify whether (YES) or not (NO) JES2 is allowed to wait on this call request. If a WAIT is necessary within this \$DOGJQE service and WAIT=NO has been specified, then the ACTION requested will be not be done. DEFER specifies that a RETURN action be automatically scheduled for a later time if it is not possible to RETURN a JQA at the current time.

**ERRET=**

Specifies a label to be branched to or a register to be branched on if a non-zero return code is returned in R15.

**OKRET=**

Specifies a label to be branched to or a register to be branched on if a zero return code is returned in R15.

**QPOST=**

Specifies the conditions under which the QPOST routine will be called. If YES is coded, a call to QPOST is made regardless of any other factors. If COND is coded, then the call is dependent upon whether the PCE has \$WAITed since FETCHing the JQA. QPOST=COND is the default. See note in #POST description for valid use.

**SPECIAL=**

Specifies whether (YES) or not (NO) the caller is willing to not get the BERT lock for the JQA. This implies that no \$WAITs will occur if a change is made. SPECIAL=YES is valid only for

ACTION=(FETCH,UPDATE) calls and must also be specified for the corresponding ACTION=RETURN call.

### **#POST=**

Specifies the conditions under which the \$#POST routine will be called. If YES is coded, a call to \$#POST is made regardless of other factors. If NO is coded, then \$#POST is not called unless some other user of the JQA, for example another element in the stack, has specified #POST=YES. #POST=NO is the default.

**Note:** QPOST and #POST are valid only if the first operand of ACTION= is RETURN or CKPT. QPOST and #POST will have an effect only when the final RETURN of an update mode JQA is made.

## **Return codes**

The following return codes (in decimal) are returned in register 15.

### **Return Code Meaning**

**0**

Processing successful. Lock was not held if ACTION=QUERYLOCK.

**NZ**

Contains the member number holding the BERT lock if ACTION=QUERYLOCK was specified.

**-1**

ACTION=QUERYLOCK was specified and lock was available, but an attempt to lock the JQE would fail because of a BERT shortage. Only returned if ACTION=(QUERYLOCK,OBTAINABLE).

**4**

Processing successful even though CBADDR had an address of a real JQE on an ACTION=CKPT call.

**8**

WAIT=NO was specified and a WAIT was required. ACTION not serviced.

**12**

WAIT=YES was specified and the JQE disappeared while waiting for the BERT lock.

On return from FETCH, CKPT, REFRESH, GETJQEADDR, FETCHNEXT, and SETACCESS type \$DOJJQE calls:

- Register 0 contains the address of the JQA returned.
- Register 1 contains the address of the JQE returned.

On return from a GETJQEADDR type \$DOJJQE call:

- Register 1 contains the address of the real JQE.

On return from a QUERYLOCK type \$DOJJQE call:

- Register 1 contains the PCE address of the member holding the lock if the lock is held by this member.

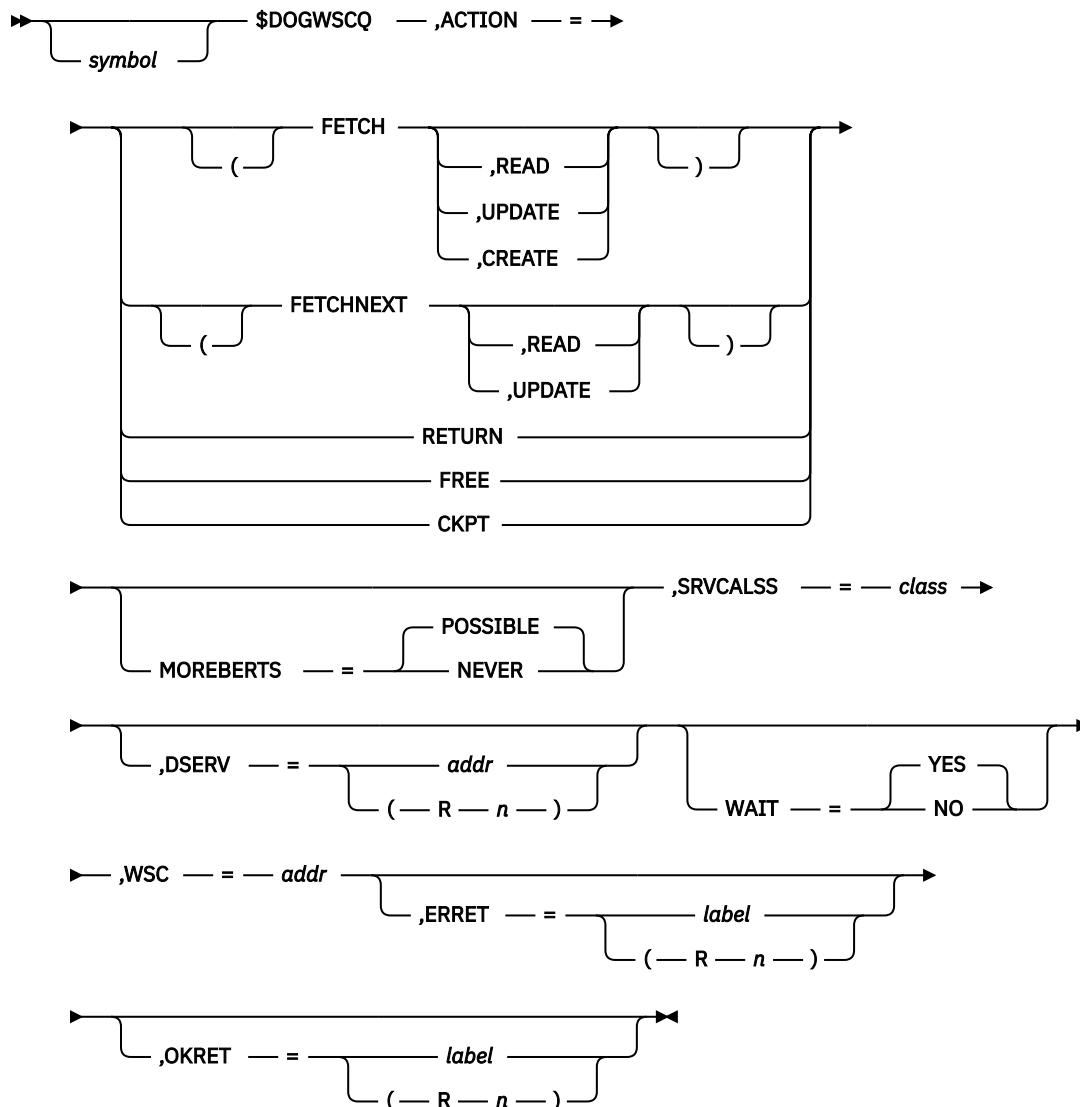
## **Environment**

- Main task (all actions are available)
- USER, SUBTASK, and FSS environments (ACTION=FETCH for READ access only)
- \$WAIT can occur.

## **\$DOGWSCQ – Deliver or get workload management (WLM) service class**

Use \$DOGWSCQ to invoke the workload manager service class (WSC) processing services to allow you to manipulate WLM service such as: get a copy of the WSC from the checkpoint or remove the WSC from the checkpoint.

## Format description



### ACTION=

The function you request of the \$DOGWSCQ service.

#### (FETCH)

Request that JES2 return the WSC for the specified service class.

#### (FETCH,READ)

Specifies that JES2 should return a read-only copy of the WSC.

#### (FETCH,UPDATE)

Specifies that JES2 should return a copy of the WSC that you can update. JES2 locks the WSC using the block extension reuse table (BERT) lock until you return the updated copy.

**Note:** UPDATE is mutually exclusive with DSERV=.

#### (FETCH,CREATE)

Specifies that JES2 should return a copy of the WSC that you can update and create one if none exists yet.

#### (FETCHNEXT)

Specifies that JES2 put the current control block into the BERT and then return the next BERT in the chain.



**Note:**

1. You must first do a FETCH before doing a FETCHNEXT.
2. If you specify FETCHNEXT you cannot also specify SRVCLASS=.

**(FETCHNEXT,READ)**

Specifies that JES2 should return a read-only copy of the WSC.

**(FETCHNEXT,UPDATE)**

Specifies that JES2 should return a copy of the WSC that you can update. JES2 locks the WSC using the block extension reuse table (BERT) lock until you return the updated copy.

**Note:** UPDATE is mutually exclusive with DSERV=.

**RETURN**

Specifies that JES2 should unlock the WSC, write the NEW values in the WSC to the checkpoint, and free the work space storage.

**FREE**

Specifies that JES2 should free the BERTs associated with this WSCQ.

**CKPT**

Specifies that JES2 should write the WSC to the checkpoint.

**Note:** WSC= is required for CKPT.

**MOREBERTS=**

Specifies the action to take for BERTs. MOREBERTS= is not valid if ACTION=RETURN or ACTION=FETCHNEXT is specified. POSSIBLE is the default.

**POSSIBLE**

Indicates that the maximum number of BERTs that might be needed are obtained before the WSC is returned.

**NEVER**

Indicates that sufficient BERTs are already assigned for the WSC to return.

**SRVCLASS=**

Specifies the service class from which JES2 obtains the WSC.

**DSERV=**

Specifies the address of the DSERV control block for the checkpoint version you request JES2 to use. If you do not specify DSERV=, JES2 uses the real CKPT.

**Note:**

1. DSERV= is not allowed in the JES2 main task environment.
2. You must specify DSERV= in User, Subtask, and FSS environments.
3. You must obtain the IAZDSERV data area, which describes the checkpoint version, prior to invoking the \$DOGWSCQ service. Use a \$DSERV GET call to do so. See [Appendix D, "Accessing checkpoint control blocks outside the JES2 main task,"](#) on page 481 for a typical coding example.

**WAIT=**

Specifies whether (YES) or not (NO) the caller can \$WAIT when the WSC is returned. WAIT= can be specified for ACTION=(FETCH,UPDATE), ACTION=(FETCH,CREATE), or ACTION=(FETCHNEXT,UPDATE). WAIT= is not valid if DSERV= is specified. YES is the default.

**YES**

Indicates that the caller can \$WAIT when the WSC is returned.

**NO**

Indicates that the caller cannot \$WAIT when the WSC is returned. If there are not enough BERTs for a maximum sized WSC to return, no WSC is obtained and the return code is set to 8.

## \$DOM

### WSC=

For an ACTION=FETCH call, JES2 obtains and returns the address of the WSC to the caller. The caller should not provide a work area on this type of call. For an ACTION=RETURN or ACTION=CKPT call, the caller provides the address of the WSC that JES2 will return.

### ERRET=

Specifies a label to be branched to or a register to be branched on if a non-zero return code is returned in R15.

### OKRET=

Specifies a label to be branched to or a register to be branched on if a zero return code is returned in R15.

## Environment

- JES2 Main task only
- FETCH and FETCHNEXT for READ access is available in User, Subtask and FSS environment
- \$WAIT can occur

## Return codes

The following return codes (in decimal) are returned in register 15.

Return Code	Meaning
-------------	---------

0	Request successful.
---	---------------------

4	Not found.
---	------------

8	WSC not returned because of insufficient BERTs
---	--

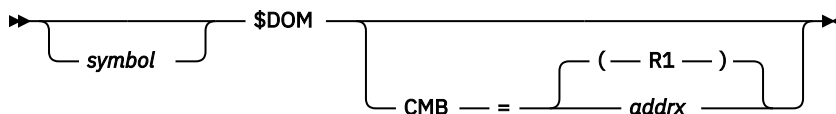
On return, register 1 contains the address of the WSC returned.

## \$DOM – Delete operator message

---

Use \$DOM to delete an operator message.

## Format description



### CMB=

Specifies the address of the command message buffer (CMB) containing the operator message to be deleted. If register notation is used the register must contain the address of the CMB before executing the \$DOM. If CMB= is not specified register 1 is assumed to contain the address of the CMB.

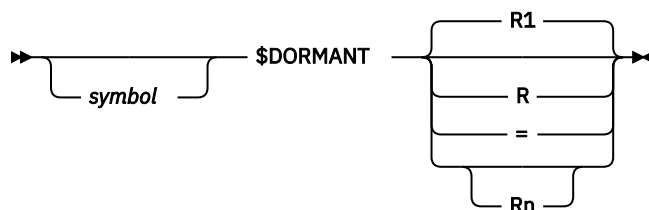
## Environment

- Main task.
- \$WAIT cannot occur.

## \$DORMANT – Specify processor is inactive

Use \$DORMANT to indicate to the JES2 dispatcher that the associated JES2 processor has completed the processing of a job or task.

### Format description



#### R

Specifies the register which is to be used by the \$DORMANT macro instruction. If R is omitted, register 1 is used.

**Note:** Do not enclose the specified register in parenthesis.



#### Attention:

The \$DORMANT macro instruction should never be used unless a corresponding \$ACTIVE macro instruction has been used for the same processor.

### Environment

- Main task.
- \$WAIT cannot occur.

## \$DSERV – Obtain or free a DSERV pointer

Use \$DSERV to request that JES2 invoke checkpoint version processing to either obtain (GET) or free (FREE) a DSERV, the parameter list used by authorized programs to request job information service from the JES2 checkpoint data space.

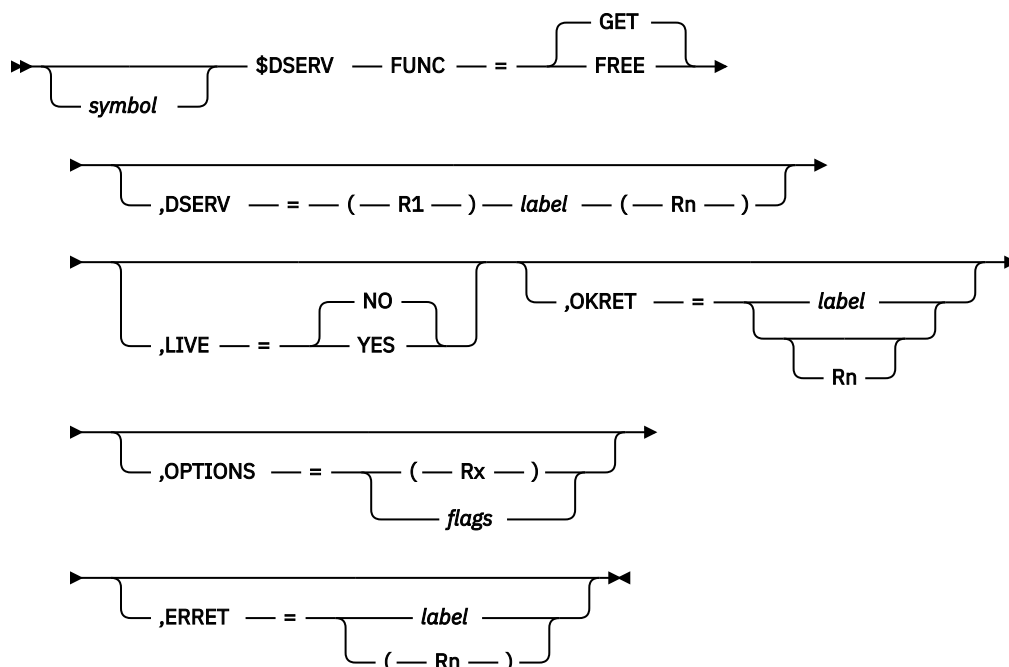
A \$DSERV call accesses the checkpoint data set through an SSI request (IEFSSREQ) to obtain or free a DSERV (IAZDSERV), the control block that describes the checkpoint version. Until you release the DSERV, JES2 holds constant the copy of the checkpoint data set associated with this DSERV call. For information about the IEFSSREQ macro, see [z/OS MVS Using the Subsystem Interface](#).

#### Note:

1. Within the invoking application or exit, it is highly recommended that you provide a corresponding FREE call for every \$DSERV GET. If you do not code the associated FREE call, JES2 does not release the version until the application address space terminates.
2. Because the number of checkpoint versions is set at the system level, holding a checkpoint level for an extended period of time can have negative performance implications for other address spaces and processes. For more information on how JES2 handles checkpoint versions, see [z/OS JES2 Initialization and Tuning Guide](#).

JES2 controls the maximum number of versions that can exist at one time based on the CKPTDEF VERSIONS= setting, and JES2 will not create a new version if the number of versions has reached the maximum. JES2 will, however, make the most recently created version available to your application. For more information on defining VERSIONS= see [z/OS JES2 Initialization and Tuning Reference](#).

## Format description



### **FUNC=**

Specifies whether JES2 is to obtain (GET) or free (FREE) a DSERV. The default is `FUNC=GET`.

### **DSERV=**

- If `FUNC=FREE`: specifies a fullword field or register containing the DSERV pointer.
- If `FUNC=GET`: specifies the register in which JES2 returns the address of the DSERV. If `DSERV=` is not specified, JES2 uses `R1`.

### **LIVE**

Specifies whether the "live" version of the checkpoint is to be returned. A live version is a pointer to a data space instance of the actual checkpoint data (shared virtual storage). Live versions are updated by the JES2 main task as you are looking at them. It is the application's responsibility to deal with data structures (in particular control block chains) that are being updated. The parameter is optional and the default is `NO`.

### **OKRET=**

Specifies a label or register that contains the address of an error routine to which JES2 branches if JES2 returns a zero return code in `R15`.

### **OPTIONS=**

Specifies option flags to be passed on a `FUNC=GET` call. The options are flags which are associated with `DSRVFLG1` in the `IAZDSERV` macro. Do not use `OPTIONS=` to request a live version (`DSRVF1LI` flag); use the `LIVE=YES` keyword. If a register is specified, the options should be in byte 7 of the 64 bit register.

### **ERRET=**

Specifies a label or register that contains the address of an error routine to which JES2 branches if JES2 returns a non-zero return code in `R15`.

## Return codes

The following return codes (in decimal) are returned in register 15.

Return Code	Meaning
-------------	---------

- 0** Processing successful (no errors)
- 4** Storage not obtained (GET processing)
- 8** CKPTVERS processing failed
- 12** Version not obtained, SSOBRETN (return code from SSOB) is greater than or equal to 4

## Environment

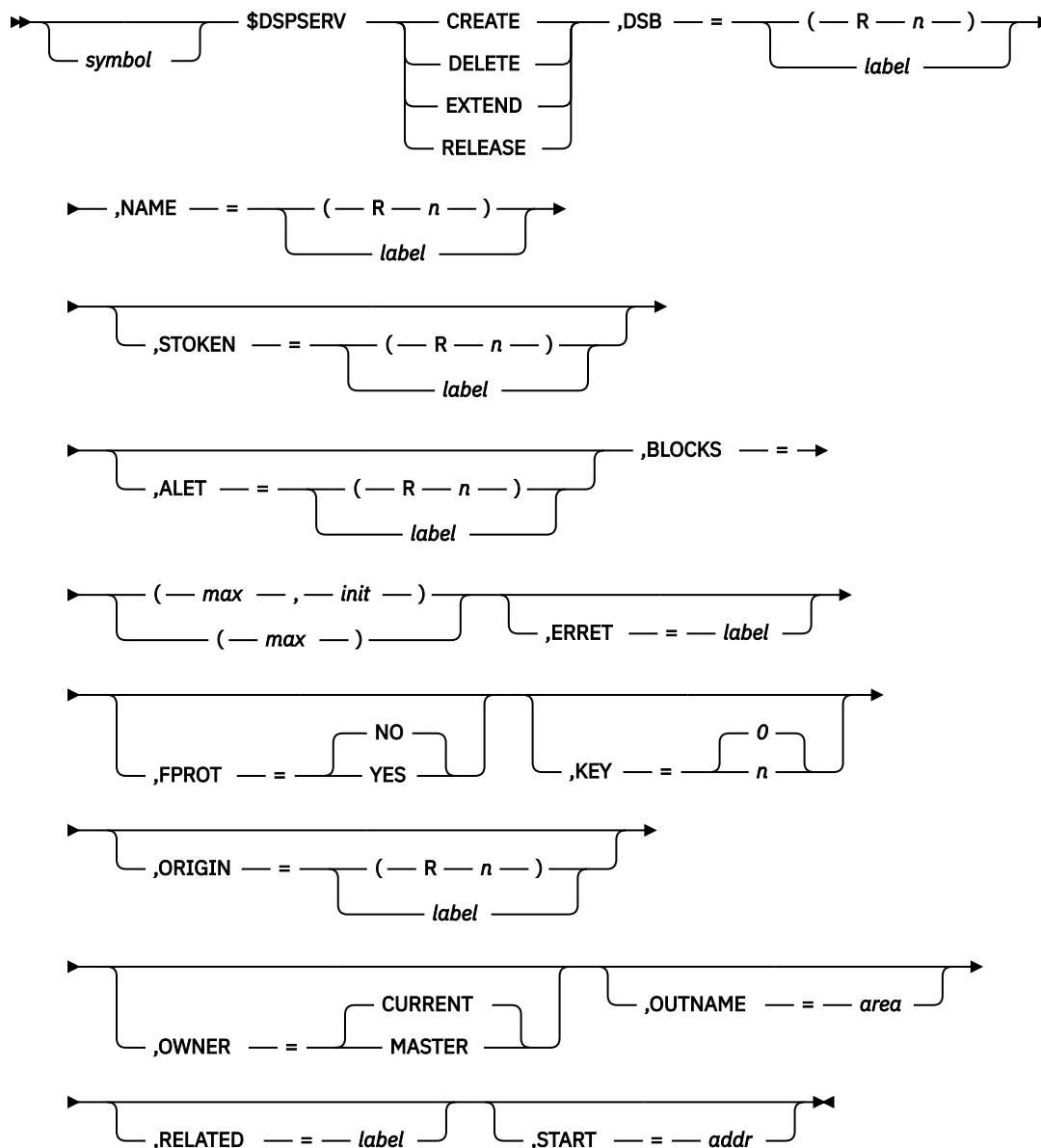
All environments except JES2 maintask.

## **\$DSPSERV – JES2 data space services**

---

Use the \$DSPSERV macro instruction to call the \$DSPSERV service routines to create or delete data spaces. See [“Data space usage” on page 479](#) for a description of data spaces.

## Format description



### CREATE | DELETE | EXTEND | RELEASE

Specifies whether the call to \$DSPSERV is to:

- CREATE a new data space  
If you specify CREATE, you must also specify BLOCKS=, NAME=, and OWNER=.
- DELETE an existing data space  
If you specify DELETE, you must also specify DSB=.
- EXTEND the size of an existing data space  
If you specify EXTEND, you must also specify BLOCKS= and DSB=.
- RELEASE page of storage.  
If you specify RELEASE, you must also specify BLOCKS=, DSB=, and START=.

If you specify CREATE, you must also specify OUTNAME=.

**Note:** You **must code** one of these functions.

**ALET=**

Specifies either a register that contains the address of the ALET of a new data space or the label of a storage area that contains the ALET of a new data space. (The ALET can alternatively be obtained through a \$ALESERV macro call.)

**BLOCKS=(max,init) | (max)**

Specifies the number (1-524288 of 4K blocks of storage to be processed for the data space. If you specify a register, the value in the register is the number of 4K blocks of storage JES2 will process.

**Note:** The upper limit, 524288 is valid for all \$DSPSERV functions, but assumes a '0 origin' data space. However, because some hardware does not support 0 origin data spaces (page at location 0 is reserved), a reliable value of 524287 is always respected.)

Register notation requires an extra set of parenthesis as exemplified below:

**Syntax for CREATE:**

```
BLOCKS=((R1),(R2))
BLOCKS=((R1),INIT)
BLOCKS=(MAX,(R2))
BLOCKS=((R1))
```

where:

max - the maximum size (in 4K blocks) to which the data space can be extended  
 init - the initial number of 4K blocks that the data space will contain.  
 If you do not specify an INIT value, it default to the MAX value.

**Syntax for EXTEND:**

```
BLOCKS=numblocks
```

where:

numblocks - the number of 4K blocks to add to the existing data space.  
 Maximum block size is limited by the \$DSPSERV CREATE,  
 BLOCK=(max) value.

**Syntax for RELEASE:**

```
BLOCKS=numblocks
```

where:

numblocks - the number of 4K blocks to be released from the existing data space.

This keyword is not valid for DELETE and if it is specified, the macro will issue an error message.

**DSB=**

Specifies the address of the data space block (DSB) for the data space to be processed (EXTEND, RELEASE, DELETE), or where to put the data space (CREATE).

You must specify a DSB= address for EXTEND, RELEASE, and DELETE; it is optional for CREATE.

**ERRET=**

Specifies the label of an error routine that is to receive control if \$DSPSERV receives a non-zero return code.

The following return codes (in decimal) are returned in register 15.

**Return Code****Meaning**

**0**

Processing successful

**8**

An ESTAE could not be established

**12**

Data space services encountered a severe error

## **\$DSPSERV**

**16**

Catastrophic recursion error; disposition of data space unknown. (Requested function might have completed.)

**40**

No ECSA storage available for a DSB

**44**

Unable to obtain working private storage

**48**

Macro was not issued with a valid function

**52**

Macro not issued with a valid CREATE,BLOCKS= specification

**56**

SRB request failed

**60**

MVS macro TCBTOKEN failed

**64**

MVS macro DSPSERV failed

**68**

MVS macro ALESERV failed

**76**

DSWA level not valid

### **FPROT=**

Specifies whether (YES) or not (NO) the data space is fetch protected on a CREATE call.

This keyword is valid only for a CREATE call.

### **KEY=**

Specifies a 1-byte value of the storage key for the data space. The key is held in bits 0-3; bits 4-7 are ignored.

KEY= is only valid on a \$DSPSERV CREATE call.

### **NAME=**

Specifies the register that contains the address of, or the name of a storage location that contains, the name of the data space to either be created or deleted. This name can be useful if you need to locate the data space in a dump taken through IPCS (interactive problem control system). Two data spaces can not have the same name.

NAME= is only valid and required on a \$DSPSERV CREATE call.

### **ORIGIN=**

Specifies the register to contain the address of, or the name of a storage area to contain the address of, the lowest address in the data space.

ORIGIN= is only valid on a CREATE call.

### **OUTNAME=**

Specifies an 8-byte area into which the name of the newly created data space is placed.

OUTNAME= is only valid on a CREATE call.

### **OWNER=**

Specifies the owner of the data space, as follows:

#### **Value**

**Data Space Owned by:**

#### **AUX**

The JES2AUX (auxiliary) address space



**CURRENT**

The task currently in control (Required if SCOPE=LOCAL)

**MASTER**

Master scheduler main task

OWNER= is required and only valid on a CREATE call.

**RELATED=**

Use this keyword for self-documentation of the macro. You can use this to keep track of the data spaces you have created or deleted. Any alphanumeric form is valid.

**START=**

Specifies the beginning address in the data space of an area to be RELEASED.

START= is required and only valid on a RELEASE call.

**STOKEN=**

Specifies either a register that contains the address of the STOKEN value of a new data space or the label of a storage area that contains the STOKEN value of the new data space.

STOKEN= is only valid on a CREATE call.

## Programming considerations

\$DSPSERV functions, CREATE and DELETE, assume serialization by the caller.

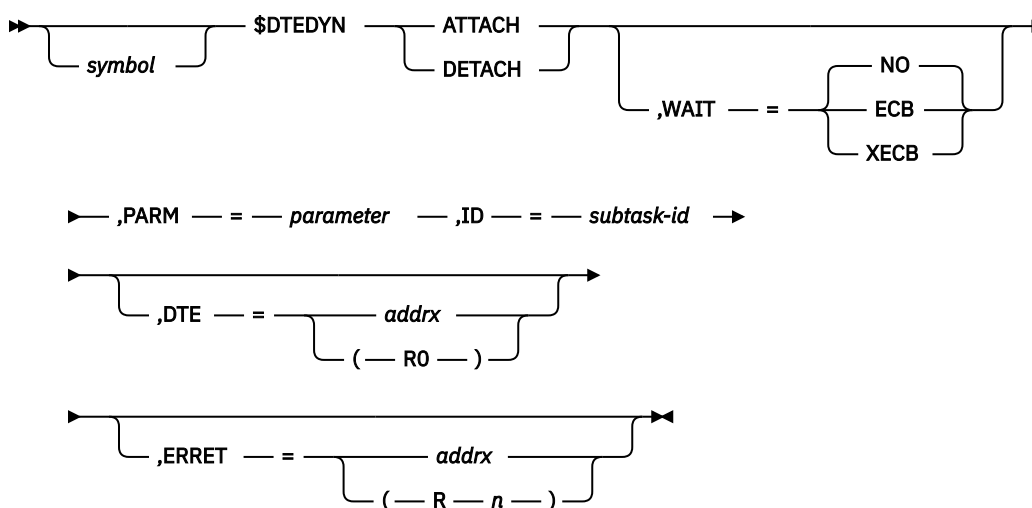
## Environment

- Main task limited (initialization and termination).
- MVS WAIT can occur during initialization and termination.
- Callers in AR ASC mode are supported. All data areas passed must be addressable in primary ASC mode.

## \$DTEDYN – Call the dynamic DTE service routines

Use the \$DTEDYN macro instruction to call the dynamic DTE service routines (\$DTEDYNA and \$DTEDYND) located in HASPDYN that handles DTE management and subtask attaches and detaches for the JES2 main task.

## Format description

**ATTACH | DETACH**

Specifies whether to call the \$DTEDYNA (ATTACH) or \$DTEDYND (DETACH) service routine.

**ATTACH**

Informs \$DTEDYNA to obtain and initialize a new DTE and to attach the subtask for the caller.

The following return codes (in decimal) are returned in register 15.

**Return Code****Meaning****0**

Processing successful

**4**

Processing failed. \$GETWORK failed to obtain the new DTE storage.

**8**

Processing failed. The MVS ATTACH macro processing returned a nonzero return code (returned to the caller of \$DTEDYN in register 1).

**DETACH**

Informs \$DTEDYND to free up the DTE and to detach the subtask.

The following return codes (in decimal) are returned in register 15.

**Return Code****Meaning****0**

Processing successful

**WAIT=**

Specifies whether \$DTEDYN should wait for subtask initialization and/or termination.

ATTACH Processing.

**ECB**

Indicates that \$DTEDYNA should wait for the subtask to post the initialization ECB.

**XECB**

Indicates that \$DTEDYNA should \$WAIT (XECB style) for the subtask to post the initialization ECB.

**NO**

Indicates that \$DTEDYNA is not to wait.

DETACH Processing

**ECB**

Indicates that \$DTEDYND should wait for MVS to post the subtask termination ECB.

**XECB**

Indicates that \$DTEDYND should \$WAIT (XECB style) for MVS to post the subtask termination ECB.

**NO**

Indicates that \$DTEDYND is not to wait.

**PARM=**

Specifies a full-word parameter to be passed to the subtask during ATTACH processing. Do not use this keyword during DETACH processing.

**ID=**

Specifies the subtask identifier as defined in the \$DTE control block. The subtask ID is passed to the \$DTEDYN service in register 1. If this keyword is not specified, an assembly error will occur.

**DTE=**

Specifies the address of the DTE to be freed by \$DTEDYND.

**Note:** This keyword must be specified for \$DTEDYN DETACH; it is not valid for \$DTEDYN ATTACH.

**ERRET=**

Specifies the address of an error routine that is to get control if an error occurs during dynamic DTE processing.

## Environment

- Main task.
- \$WAIT can occur in JES2 main task.
- MVS WAIT can occur during initialization and termination.

**Note:** The subtask could be dispatched before \$DTEDYN processing completes.

## \$DTETAB – Build and map the DTE tables

Use the \$DTETAB macro instruction to build the DTE tables accessed by the \$GETABLE service.

Use \$DTETAB to map and generate DTE table entries.

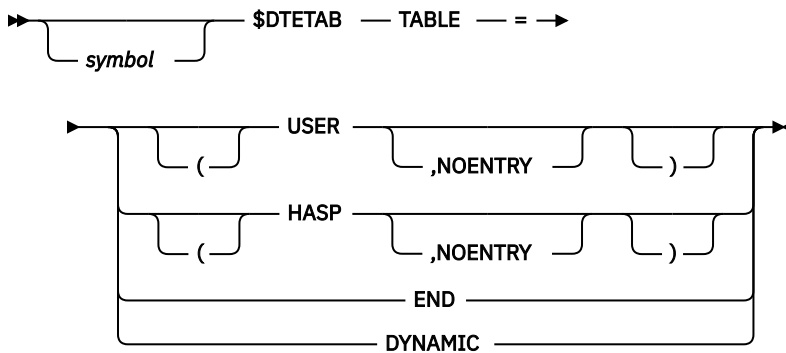
\$DTETAB entries are used to define the start of a user table (\$DTETAB TABLE=USER...) or a JES2 table (\$DTETAB TABLE=HASP...), the end of a table (\$DTETAB TABLE=END) or an entry in a table (\$DTETAB NAME=ANANIA...).

**Note:** The format description that follows breaks the macro into a **boundary form** (the form that starts or ends a table) and an **entry form** (the form that defines each table entry).

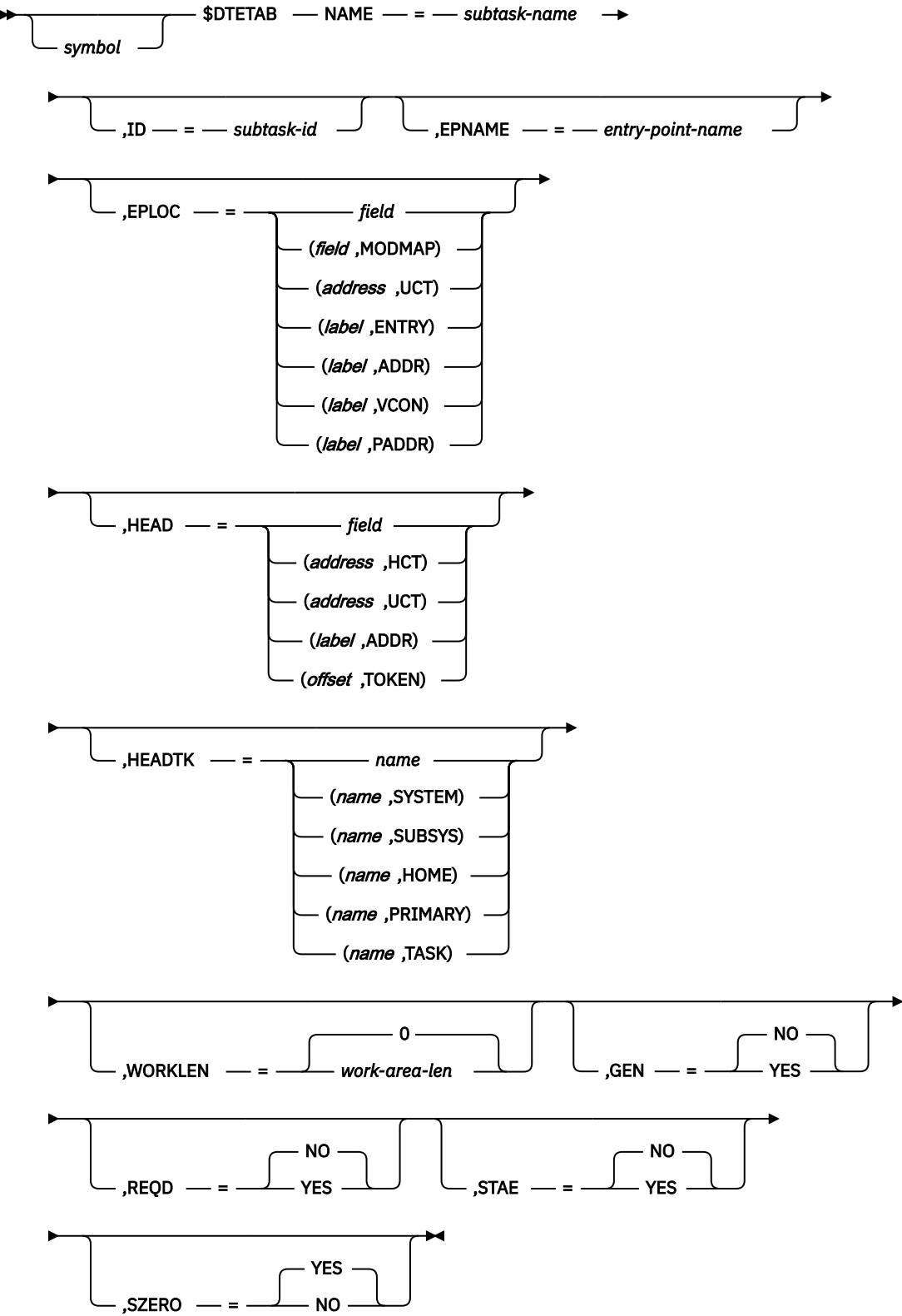
## Format description

The following formats apply:

### Boundary form



Entry form



**NAME=**  
Specifies the subtask name that HASP messages use to identify the subtask to the operator.

**ID=**  
Specifies the subtask identifier used in the \$DTE DSECT.

**EPNAME=**

Specifies the entry point name used by \$DTEDYNA for the MVS IDENTIFY macro call.

**EPLOC=**

Specifies the offset into the specified control block and, optionally, the control block name from which the entry point address is obtained, entry, address and VCON. The control block name defaults to either MODMAP (if this macro is used to build a HASP DTE table) or UCT (if this macro is used to build a USER DTE table). If the 'control block name' is specified here it overrides either default value.

Specify EPLOC as follows:

**field**

Specifies a MODMAP field if this is the Hasp table or a UCT field if this is the USER TABLE.

**(field,MODMAP)**

Explicitly specifies a MODMAP field.

**(field,UCT)**

Explicitly specifies a UCT field.

**(label,ENTRY)**

Indicates that 'label' is the name of the subtask entry point routine defined using the \$ENTRY.

**(label,ADDR)**

Specifies a label in the current module .

**(label,VCON)**

Specifies a label in the external module.

**(label,PADDR)**

Specifies a routine whose address is in the \$PADDR.

**HEAD=**

Specifies the offset, control block name, address and token of the subtask type chain head. The control block name defaults to either HCT (if this macro is used to build a HASP DTE table) or UCT (if this macro is used to build a USER DTE table).

Specify HEAD as follows:

**field**

Specifies a HCT field if this is the HASP table or a UCT field if this is the USER table.

**(field,HCT)**

Explicitly specifies a HCT field.

**(field,UCT)**

Explicitly specifies a UCT field.

**(label,ADDR)**

Specifies a label in the current module.

**(offset,TOKEN)**

Indicates that HEAD field CB address is in MVS name/token pair. HEADTK= must be specified.

**HEADTK=**

Specifies the NAME associated with a name/token pair (created using the IEANTCR callable service). The name/token pair contains the address of the control block that contains the HEAD field. NAMES can be up to 16 bytes long (and must match the name specified on a IEANTCR call). The first 4 bytes of the token are assumed to be the control block address. The remaining 12 bytes of the token are not used.

The control block address (from the token) is combined with the offset specified on HEAD= to determine the DTE pointer. HEADTK= is required, and only allowed, if HEAD=(offset,TOKEN) was specified.

NAME/TOKEN pairs can be created at various levels. JES2 supports tokens at the SYSTEM, TASK, or HOME level.

The second operand on HEADTK= specifies the level passed on the IEANTRT service call.

**HEADTK=(name,SYSTEM)**

Indicates SYSTEM level.

**HEADTK=(name,SUBSYS)**

Indicates SYSTEM level with the last 4 bytes of the 16-byte name replaced by the subsystem id.

**HEADTK=(name,HOME)**

Indicates HOME level.

**HEADTK=(name,PRIMARY)**

Indicates PRIMARY.

**HEADTK=(name,TASK)**

Indicates TASK level.

**HEADTK=name**

Defaults to TASK level .

**WORKLEN=**

Specifies the length of the subtask work area extension. If specified, this length is added to the DTE length (DTELEN) when \$DTEZYNA obtains DTE storage. WORKLEN=0 (no storage) is the default.

**GEN=**

Specifies whether (YES) or not (NO) the subtask is ATTACHed during IRMVS processing. YES indicates that the DTE1FLAG flag, DTE1AUTO is set on to indicate subtask ATTACH. GEN=NO is the default.

**REQD=**

Specifies whether this subtask is essential to system operation and what needs to be done when the subtask abnormally terminates.

**YES**

A non-recoverable abend is issued by the subtask (that is, CALLRTM RETRY=NO,...) that will cause the JES2 maintask to be terminated with an abend \$Z03.

**TYPE**

A non-recoverable abend will only be issued by the last remaining subtask of this type to cause the JES2 maintask to be terminated with an abend \$Z03.

**NO**

The subtask is not required.

**STAE=**

Specifies whether (YES) or not (NO) the subtask is DETACHed if STAE is specified on the DETACH macro. YES indicates that the DTE1FLAG flag, DTE1STAE is set on to indicate subtask DETACH. STAE=NO is the default.

**SZERO=**

Specifies whether (YES) or not (NO) the DTEFLAG1 flag, DTESUB0, is set on at the MVS ATTACH call. YES indicates that the DTEFLAG1 flag, DTE1SUB0 is set on. SZERO=YES is the default.

**TABLE=**

Specifies either the beginning of a USER or HASP DTE table (TABLE=USER or TABLE=HASP, respectively) or the end of a previously specified table (TABLE=END). The NOENTRY operand indicates that the ENTRY statement will not be generated for this specific DTE table. If TABLE= is specified, all other keywords on this macro are ignored. DYNAMIC specifies that this is a dynamic table.

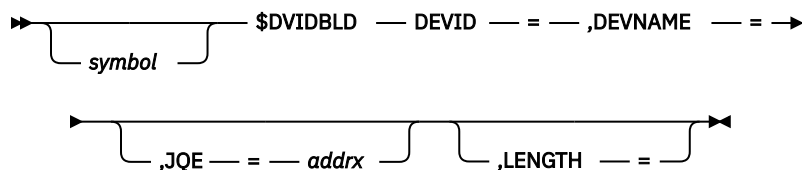
## Environment

- JES2 main task or during JES2 initialization or termination.
- \$WAIT is not applicable – this macro generates a DSECT or a static table entry; it does not generate executable code.

## \$DVIDBLD – Build a device name from a device identifier

Use the \$DVIDBLD macro to convert a binary device identifier (devid) into a character device name

## Format description



### DEVID=

Specifies a 3-byte field containing the device id. (See field name, DCTDEVID, in the DCT for possible values).

### DEVNAME=

The field in which JES2 will return the character device name. This field must be a minimum of 8 bytes.

### JQE=

Specifies the address of a JQE to determine if this is an INTRDR, STCINRDR, or TSUINRDR.

**Note:** Be aware, if you specify JQE=, this macro alters the contents of register 2.

### LENGTH=

Specifies the length of the DEVNAME= field. If you do not specify this length, it defaults to the assembled length of the field.

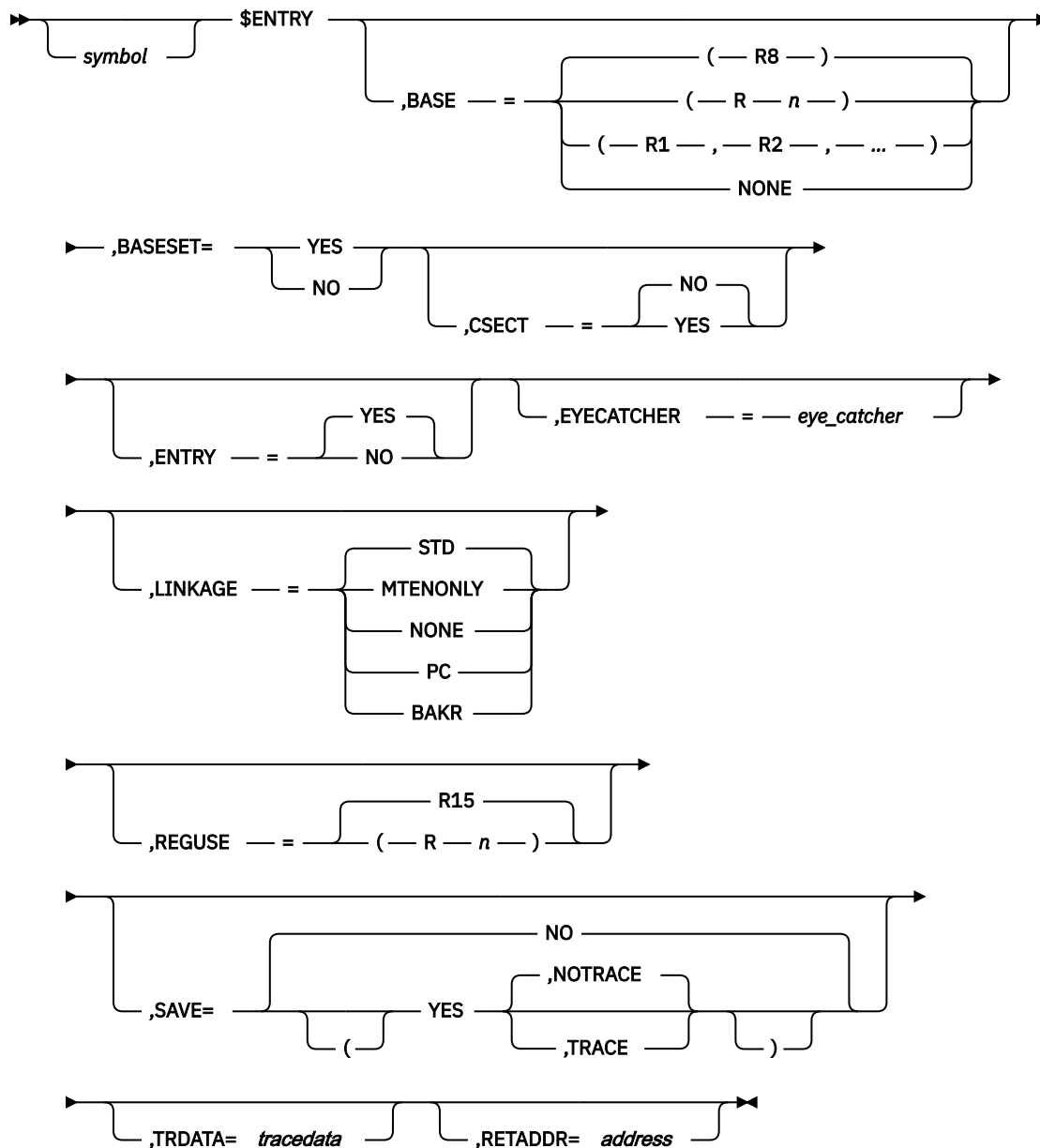
## Environment

- JES2 main task or user address space
- 31-bit addressing mode
- Caller can be in AR ASC mode
- MVS WAIT OR JES2 \$WAIT cannot occur

## \$ENTRY – Provide entry to JES2 assembly module

Use \$ENTRY to identify, to JES2, entry points in an assembly module such as an installation-provided exit routine. Each exit routine must use \$ENTRY to define all of the routine's entry points.

## Format description



### symbol

Specifies an entry point name. You must code this parameter. If you code `ENTRY=YES`, `$ENTRY` creates an entry in the assembler ESD for *symbol*.

**Note:** Be certain the value specified **is different** from any symbol specified on other `$ENTRY` or `$MODULE` statements.

### BASE=

Specifies the registers that provide addressability to the entry point. If two or more registers are specified, they must be separated by commas and enclosed in parentheses. If `BASE` is not specified, `BASE=R8` is assumed. A `USING` statement is generated using the specified registers. If `BASE=NONE` is specified then no code base register is assumed or set up by `$ENTRY`.

`$ENTRY` does not load the base registers unless `SAVE=YES` is also specified. If `SAVE=NO` is specified (or defaulted), then it is your responsibility to ensure that each base register is loaded with the entry point address.



**CSECT=**

Specifies whether a CSECT statement should be generated. If you specify CSECT=YES, the assembled module will include the JES2 version number and the date and time of the assembly. \$ENTRY also creates an entry in the MIT entry table.

IBM suggests that you do not define multiple CSECTs in any JES2 assembly modules including installation exit routines. The default is NO.

**ENTRY=**

Specifies whether \$ENTRY is to generate an assembler ENTRY statement for the symbol parameter that is coded on the \$ENTRY macro.

**YES**

\$ENTRY is to generate an ENTRY statement and place an entry in the MIT entry table.

**NO**

\$ENTRY is to place an entry in the MIT entry table but is not to generate an ENTRY statement.

If you code the CSECT= parameter, omit the ENTRY= parameter.

**EYECATCHER=**

Specifies an *eyecatcher* that \$ENTRY is to include in the assembled module. An *eyecatcher* is a character string, typically the module name or entry point name, that makes it easier to identify the module in a storage dump. The eyecatcher appears near the module entry point.

You can specify any value that is valid as an operand of an assembler DC instruction. For example:

```
...,EYECATCHER=C'A_STRING',... generates DC C'A_STRING'
...,EYECATCHER=CL20'MODULENAME',... generates DC CL20'MODULENAME'
```

If you specify LINKAGE=MTEONLY or LINKAGE=NONE, omit the EYECATCHER= parameter.

**LINKAGE=**

Specifies the type of entry point environment \$ENTRY is to generate and causes JES2 to build an entry (MTE) in the module information table (MIT). The default is based on the routine name and the linkage that would be dictated by the field name in the CADDR or PADDR.

**MTEONLY**

\$ENTRY generates a USING instruction for the symbol parameter. The register specified on the USING Instruction is the register specified on BASE= or the default, register 8. Because \$ENTRY does not define the symbol parameter to the assembler, your program must do so.

**NONE**

\$ENTRY defines the symbol parameter to the assembler as a label and generates a USING instruction for that label. The register specified on the USING Instruction is the register specified on BASE= or the default, register 8.

**PC**

Specifies that the programs that invoke this entry point do so by using a stacking PC instruction.

\$ENTRY defines the symbol parameter to the assembler as a label and generates a USING instruction for that label. The register specified on the USING Instruction is the register specified on BASE= or the default, register 8. \$ENTRY uses the register specified on REGUSE= or the default, register 15, to establish initial addressability.

**STD**

Specifies that the programs that invoke this entry point do so by using a branch linkage. The register identified by the REGUSE= parameter, or register 15 when you omit REGUSE=, must contain the entry point address.

\$ENTRY defines the symbol parameter to the assembler as a label and generates a USING instruction for that label. The register specified on the USING Instruction is the register specified on BASE= or the default, register 8.

### **BAKR**

Specifies that the programs that invoke this entry point do so by using a branch and stack linkage. The register identified by the REGUSE= parameter, or register 15 when you omit REGUSE=, must contain the entry point address.

\$ENTRY defines the symbol parameter to the assembler as a label and generates a USING instruction for that label. The register specified on the USING Instruction is the register specified on BASE= or the default, register 8.

### **REGUSE=**

Specifies either a register that contains the entry point address or a register that \$ENTRY can use to establish initial addressability. The default is register 15.

- If you omit the LINKAGE= parameter or code LINKAGE=STD, \$ENTRY requires that a register contain the entry point address. You code the REGUSE= parameter to identify that register. If you omit REGUSE=, register 15 must contain the entry point address.
- If you code LINKAGE=PC, \$ENTRY uses the register specified by REGUSE=. If you omit REGUSE=, \$ENTRY uses register 15.

### **SAVE=**

Specifies whether (YES) or not (NO) a \$SAVE should be generated as part of the linkage. An optional second parameter specifies whether (TRACE) or not (NOTRACE) the \$SAVE should be traced. SAVE=NO is the default.

**Note:** This keyword is only valid for LINKAGE=STD.

### **SETBASE=**

Specifies whether (YES) or not (NO) the base registers specified on BASE= are to be set. If YES is specified then the value for the first base register is assumed to be in REGUSE. For LINKAGE=PC, the macro sets the address of label in REGUSE. The default for this keyword is YES if LINKAGE=PC or SAVE=YES was specified and NO otherwise.

**Note:** This keyword is only valid with LINKAGE=STD or LINKAGE=PC.

### **TRDATA**

Trace data to be passed to the \$SAVE macro. See \$SAVE for syntax. Only valid if \$SAVE was requested.

### **RETADDR=**

Specifies a register into which the return address should be restored. RETADDR= is valid only when LINKAGE=PC or LINKAGE=BAKR is specified.

## **Environment**

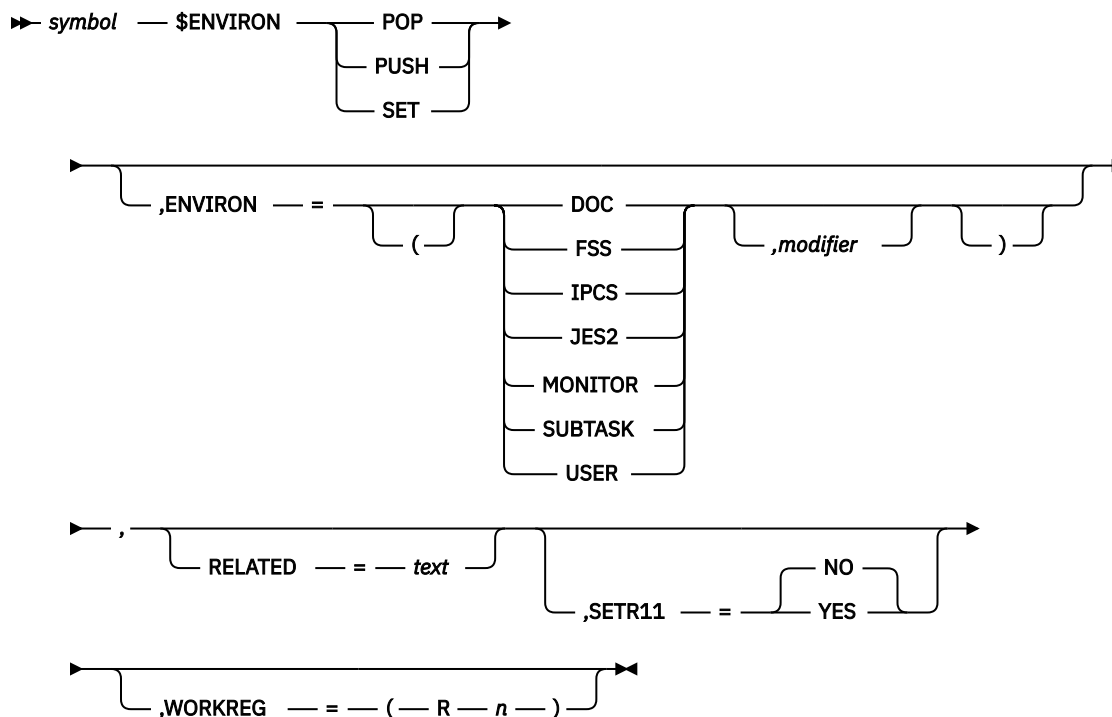
- JES2 Main task, JES2 subtask, user environment or FSS environment.
- \$WAIT not applicable.

## **\$ENVIRON – Set assembly environment**

---

Use \$ENVIRON to set the assembly environment for a JES2 module.

## Format description



### POP | PUSH | SET

Specifies the purpose of this \$ENVIRON call as follows:

#### POP

Indicates that JES2 is to restore the prior (PUSHed) assembly environment.

#### PUSH

Indicates that JES2 is to set a new assembly environment and save the old assembly environment.

#### SET

Indicates that JES2 is to modify the assembly environment.

### ENVIRON=(DOC | FSS | IPCS | JES2 | SUBTASK | USER [,modifier])

Specifies the assembly environment JES2 sets as follows:

#### DOC

Documentation-only environment (HASPD0C)

#### FSS

Functional subsystem (HASPFSM)

#### IPCS

Interactive problem control system (IPCS)

#### JES2

JES2 main task

#### MONITOR

JES2 monitor address space

#### SUBTASK

JES2 subtask

#### USER

User environment

#### modifier

This second, optional specification, can be added to any ENVIRON= specification as a modifier for the environment. For example, ENVIRON=(JES2,INIT) - INIT indicating initialization or TERM

## **\$ERROR**

indicating termination. ENVIRON=(USER,ANY) indicates that the special (USER,ANY) environment is to be used. However, *modifier* is only valid for SET and PUSH calls.

### **RELATED=**

Specifies a related \$ENVIRON service call. This parameter is for your documentation purposes only and not checked by JES2.

### **SETR11=YES | NO**

Specifies whether (YES) or not (NO) JES2 will set the address of the environment-related control block type (for example, HCCT for USER or HCT for JES2) in R11. SETR11=NO is the default.

### **WORKREG=**

Specifies a work register (R1-R10 or R12-R15) JES2 in the use when computing the address placed in R11.

## **Environment**

- Any assembly environment.

## **\$ERROR – Indicate catastrophic error**

---

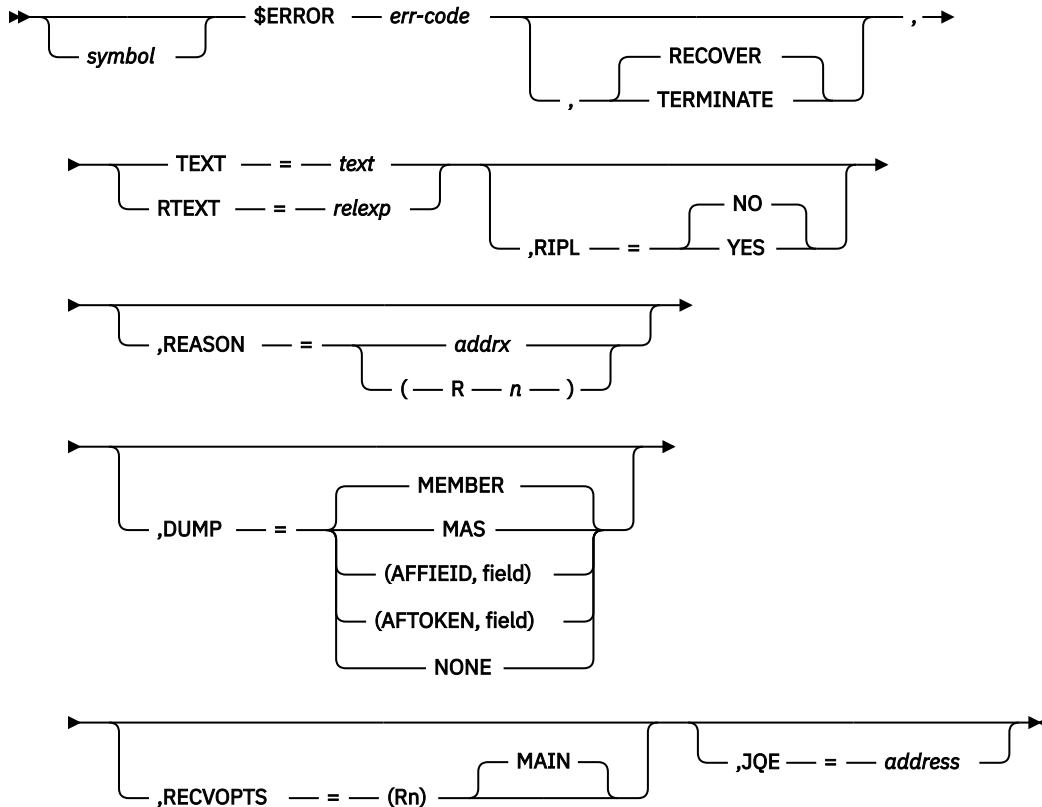
Use \$ERROR to indicate that a catastrophic error has occurred and to abend the JES2 main task, a task executing in the user address space, or an FSS address space.

In the JES2 main task, \$ERROR causes a JES2 ESTAE routine to display a \$HASP095 before receiving control on the ABEND.

### **Note:**

1. The assembly environment dictates the parameters required for this macro. The results of the macro depends on the execution environment.
2. The \$ERROR macro requires that you also issue \$MODULE with \$ERPL mapping specified.

## Format description



### symbol

A 1- to 4-character symbol. If `$ERROR` is issued from either the JES2 (main task) environment or the FSS environment, *symbol* is required. In the user environment, *symbol* is optional.

### err-code

Specifies the JES2 catastrophic error code. It is a 1- to 4-character symbol (usually a letter and two digits) preceded by a dollar sign (\$) when printed as the error code in the message. The err-code is ignored in the user environment.

### option-code

Specifies whether recovery should take place. This can be coded as follows:

#### RECOVER (default)

Recovery is to be attempted; that is, it is possible to recover from this error.

#### TERMINATE

Recovery is not to take place and an abend is to occur; that is, it is impossible to recover from this error.

### TEXT=

Specifies the text to be included in the catastrophic error message. A DC statement is generated for this character string. If you specify a character string of greater than 42 characters, JES2 will truncate it to 42 characters. This text is used in the content of the `$HASPO88` message and is required in the user environment.

Specify the text enclosed within single quotation marks or as (R0) if R0 has the address of the text string in the form: `AL1(length),C'text'`. Use (R0) in the JES2 environment only.

### RTEXT=

Specifies the symbol used on another `$ERROR` macro instruction invocation from which the text for this catastrophic error message is to be taken. This is used when there is an existing `$ERROR` macro instruction invocation with the desired text. This is ignored in the user environment.

## **\$ERROR**

### **RIPL=**

Specifies whether the system needs an IPL to recover from this error. This is ignored except in the main task environment. This can be coded as follows:

#### **YES**

The system needs an IPL.

#### **NO**

The system does not need an IPL.

### **REASON=**

Indicates a value that is only used in the user environment and specifies the reason code that appears in the \$HASP095 error message.

#### **addrx**

Indicates the address of a fullword field that contains the reason code.

#### **Rn**

Indicates a register that contains the reason code.

### **DUMP=**

Indicates whether all members of the MAS are to be dumped or this member or any associated members to be dumped. This keyword can also be used to suppress a dump.

This keyword is only valid in the JES2 environment. The default is DUMP=MEMBER. RECVOPTS does apply.

Specify one of the following:

- DUMP=MAS
- DUMP=MEMBER
- DUMP=(AFFIELD,field)
- DUMP=(AFTOKEN,field)
- DUMP=NONE

In the DUMP= text,

#### **MAS**

Specifies all members of the MAS are to be dumped.

#### **MEMBER**

Specifies this member to be dumped.

#### **AFFIELD**

Specifies associated members to be dumped. The second operand is the affinity mask field having members to be dumped.

#### **AFTOKEN**

Specifies associated members to be dumped. The second operand is the affinity token having members to be dumped.

#### **field**

Specifies affinity field containing members to be dumped, when AFFIELD is specified as the first operand. Specifies affinity token containing members to be dumped, when AFTOKEN is specified as first operand.

#### **NONE**

Specifies that no DUMP is to be taken and only LOGREC records are generated.

### **RECVOPTS=**

Specifies the RECVOPTS option to be used for this \$ERROR (JES2 main task). The value can be a string constant or a register (specified in parentheses). If not specified, the default is MAIN.

### **JQE=**

Indicates the JQE that JES2 reports in message \$HASP088 JES2 ABEND ANALYSIS. The default is the contents of the PCEJQE.

This keyword is valid in the JES2 environment only.

## Environment

- JES2 main task, JES2 subtask, FSS, or user environment.
- Will ABEND with MVS system code X'02D'.
- Will ABEND with X'0F7' in the user environment.

## \$ESTAE – JES2 error recovery environment

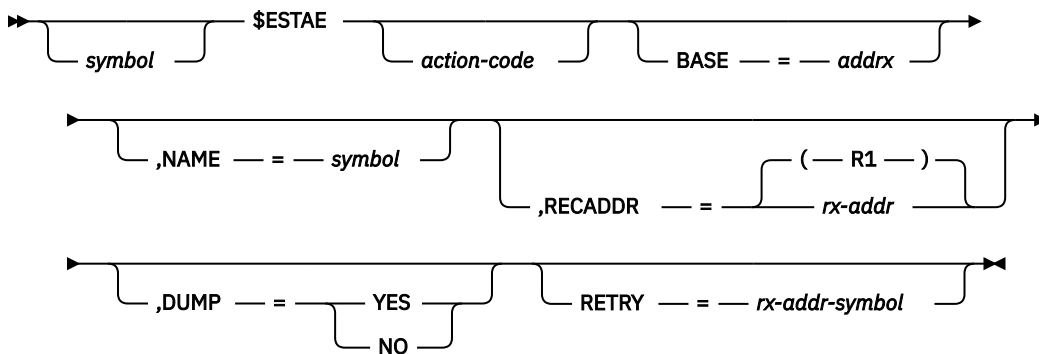
Use \$ESTAE to generate the calling sequence to one of several JES2 recovery service routines for creating, replacing or cancelling the current error recovery environment. Each error recovery environment specifies an error recovery routine that will gain control for a JES2 detected error. Each error recovery routine is represented by a processor recovery element (PRE) in the main task environment, or by a TCB recovery element extension (TRX) in the user environment.

Use this macro to:

- Create a new JES2 error recovery environment by establishing another recovery routine
- Replace the current error recovery routine with a different routine
- Cancel the accessibility of the current error recovery routine

If you issue a \$ESTAE macro instruction in the main task environment with a recovery address (RECADDR=) specified within code that is logically bracketed by \$SAVE and \$RETURN macros, the PRE created is canceled automatically in \$RETURN processing.

## Format description



### action-code

Specifies if the current recovery environment is to be canceled or replaced as follows:

#### CANCEL

Cancel the current recovery environment.

#### REPLACE

Replace the current recovery environment with a new one. REPLACE can only be specified in the main task environment.

If you omit this operand, a new recovery environment is created and stacked LIFO on the appropriate stack.

### BASE=

Specifies the address of the code base to be associated with this recovery environment. This address is placed in the PRE for use by the recovery routine. This keyword should be coded only if used by the specified recovery routine. BASE cannot be specified in the user environment.

**NAME=**

Specifies the 8-character identifier to be associated with the PRE created when you have specified either REPLACE or nothing as the first positional operand in the main task environment. If you omit this parameter, the identifier will default to the label of the \$ESTAE macro call. If you have no label specified, it will default to a system-generated identifier. NAME= cannot be specified in the user environment.

**RECADDR=**

Specifies the address of the recovery routine to gain control if JES2 detects an error.

Code this keyword, based on the environment in which you are running.

- Main task environment

In the main task environment, this keyword is optional. This can be specified as an rx-addr expression or using register notation (R2 through R12). If you do not specify this keyword, register 1 (the default) must contain the address of the recovery routine. If you specify the address as 0, recovery is suspended.

Your recovery routine should use the \$SETRP macro to request a retry or to percolate the error before returning to the caller.

- User environment

In the user task environment, this keyword is required. Specify this keyword using an A-type address.

If the RETRY keyword is also coded, the recovery routine receives control from JES2's user environment ESTAE. Your recovery routine cannot use the SETRP or \$SETRP macros to request a retry or to percolate the error. After your recovery routine returns to its caller, JES2 decides whether to retry or to percolate the error.

If the RETRY keyword is not coded, the recovery routine receives control through an MVS SETRP request from the user environment ESTAE. The recovery routine can either resume processing within the main routine, or it can return to the main routine's caller.

**RETRY=**

Specifies the address of a retry routine to be associated with this \$ESTAE in the main task or user environment.

Code this keyword, based on the environment in which you are running.

- Main task environment

In the main task environment, this keyword is optional. This can be specified as an rx-addr expression or using register notation (R2 through R12).

This address is placed in field PRERESUM in the PRE for use by the recovery routine. The recovery routine should use contents of field PRERESUM when designating a retry address, using the \$SETRP macro.

- User environment

This parameter is optional. Specify this keyword using an A-type address. JES2 gives control to the retry routine through an MVS SETRP request.

**DUMP=**

Indicates whether a dump should be captured. YES is the default. This parameter is only valid in the user environment.

**Note:**

1. If you code either CANCEL or REPLACE on the \$ESTAE macro, there must be a current PRE at the current save area level or JES2 catastrophic error \$ER1 is issued and JES2 terminates.
2. In the main task environment, \$ESTAE assumes addressability to the error recovery area (ERA) that is associated with the error that caused the recovery routine to be entered. Therefore, be certain to add the \$ERA DSECT to the \$MODULE macro for any routine for which you provide error recovery.



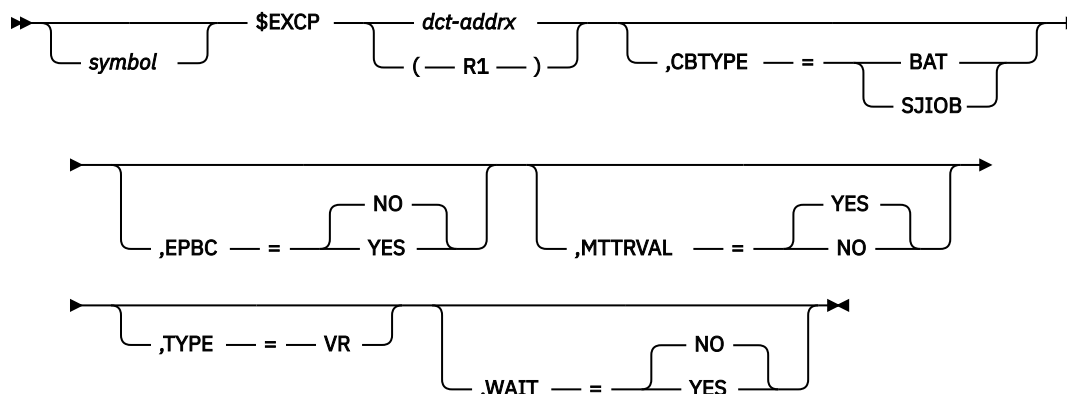
## Environment

- User task.
- Main task.
- \$WAIT cannot occur.

## \$EXCP – Execute JES2 channel program

Use \$EXCP to initiate JES2 input/output activity.

### Format description



#### dct

Specifies either the address of a pointer to a device control table (DCT) or the address of a DCT. The DCT represents a device on which input/output activity is to be initiated. If *dct* is written as an address, it represents the address of a fullword which contains the address of the DCT. If *dct* is written using register notation (either regular or special register notation), it represents the address of the DCT. If register notation is used, the address must have been loaded into the designated register before the execution of this macro instruction.

This parameter is valid only when this macro is called in the JES2 main task environment.

#### CBTYPE=

Specifies the type of the JES2 control block that contains the IOB. This parameter is only valid in the user environment and is required when the macro is called from that environment.

#### BAT

indicates a buffer auxiliary table

#### SJIOB

indicates a subsystem job input/output block

**Note:** In the user environment, issue a USING for HCCT before invoking this macro. Then, if you specify:

- CBTYPE=BAT, issue a USING for BAT before invoking this macro
- CBTYPE=SJIOB, issue a USING for SJIOB before invoking this macro

#### EPBC=

Specifies whether JES2 uses EXCP or TCBEXCP. TCBEXCP allows the I/O to be associated with a TCB other than the issuer of the service. EPBC=YES is only supported for CBTYPE=BAT.

This operand is optional, defaults to NO.

## \$EXIT

### MTTRVAL=

Specifies whether (YES) or not (NO) a \$CALL to MTTRVAL is needed to validate the module track track record (MTTR) in the DCT. If you set MTTRVAL=NO, thereby overriding JES2's verify call when performing this \$EXCP call, be certain that you have previously issued a \$CALL to verify the MTTR.

### TYPE=VR

Specifies that I/O is to be initiated through the EXCPVR macro instruction. If this parameter is omitted, EXCP is used.

This parameter is valid only when this macro is called in the JES2 main task environment.

### WAIT=

Specifies whether the \$EXCP service routine is to cause the routine issuing this macro instruction to wait (\$WAIT IO) until the I/O operation has been completed (WAIT=YES), or is to return control when the request has been scheduled (WAIT=NO or parameter omitted.)

If WAIT=YES is specified, the service routine exits after normal I/O completion. If any I/O error is detected, the service routine issues the \$IOERROR macro instruction, which issues the JES2 I/O error message, \$HASPO94, then returns control to the \$EXCP issuer. On return, register 1 points to the I/O buffer, and the condition code mask is set as follows:

#### CC=1

The I/O completed without error.

#### CC=4

Permanent I/O error was encountered.

This parameter is valid only when this macro is called in the JES2 main task environment.

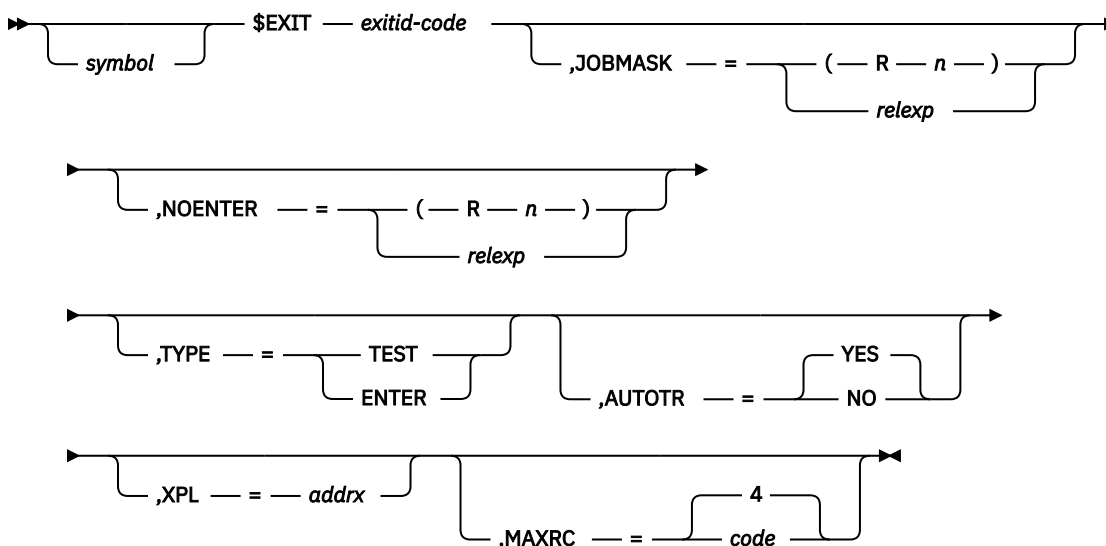
## Environment

- Main task or user environment.
- \$WAIT can occur (if you code WAIT=YES on the macro).

## \$EXIT – Provide exit point

Use \$EXIT to establish an exit point in an assembly module. The assembly environment active at the time of the \$EXIT invocation determines the exit effector that JES2 will use and the execution environment that JES2 assumes for the exit routines it will call.

## Format description



**symbol**

Although a label for this macro instruction is not required, it is highly recommended for tracing purposes.

**exitid-code**

Specifies the numeric id (0-255) of this \$EXIT macro.

**JOBMASK=**

Specifies the address, or a register that contains the address, of a 256-bit job exit mask in the job control table, of which each bit corresponds to an exit identification number; bit 0 corresponds to Exit 0, bit 1 corresponds to Exit 1, bit 2 to Exit 2, and so on. (This means, of course, that bit 2 corresponding to Exit 2 is really the third bit in the mask, and so on.) Initially, when the JCT is created, all the bits in the job exit mask are set to one. Use this operand only if the exit point is job-related, because the mask is used to determine whether the exit should be taken for a given job.

**NOENTER=**

Specifies a label to be branched to or a register to be branched on if the exit is not invoked (either because the bit for the exit is not on in the field specified in JOBMASK or because the exit is inactive). If you code TYPE=ENTER, do not code NOENTER.

**TYPE=**

Specifies how the exit effector is to treat this exit point.

If this parameter is omitted, the status of the exit point is to be determined and, if the exit is enabled, the exit effector is called to invoke the appropriate installation exit routines.

**TEST**

The exit effector tests the status of the exit point, and the exit effector sets a condition code as follows:

**CC=0**

Either the specific job mask bit for this exit is 0 or the exit id is not enabled (that is, no exit routines are to be called).

**CC=1**

The exit id is enabled but tracing is disabled.

**CC=2**

Both the exit id and tracing are active.

**ENTER**

The exit routine is to be entered through the exit effector without checking the status of the exit point. A \$EXIT macro instruction should be coded with TYPE=TEST to confirm exit point status before coding a \$EXIT macro with TYPE=ENTER.

**AUTOTR=**

Specifies whether tracing for this exit point is to be automatically provided by the exit effector. Possible values are as follows:

**YES**

YES specifies that tracing occurs if trace ID 13 is turned on (through a \$TRACE command), and either the EXIT $nnn$  initialization statement specified TRACE=YES or the operator has entered a \$T EXIT $nnn$ , TRACE=Y command for this exit point.

**NO**

No tracing takes place.

**XPL**

Address of the XPL to be passed to the exit. If specified in a register, the XPL is passed in that register to the exit. If specified in a field, it is loaded into R1.

**MAXRC=**

Specifies the maximum acceptable return code to be set by the installation exit routines. If this parameter is omitted, the default is 4.

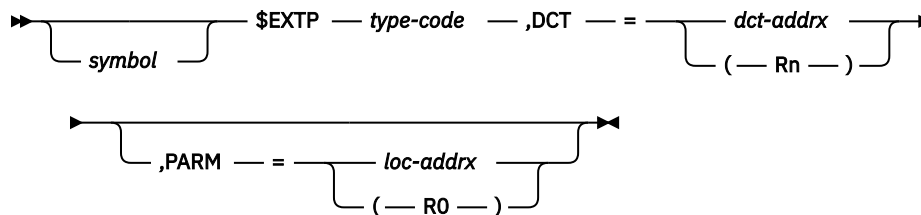
## Environment

- Main task, subtask, user, or FSS address space.
- \$WAIT can occur if an installation exit routine issues a \$WAIT or invokes a routine that issues a \$WAIT.

## \$EXTP – Initiate remote terminal input/output operation

Use \$EXTP to initiate a remote terminal or network device input/output action or operation.

### Format description



#### type

Specifies the type of operation as follows:

##### **OPEN**

Initiate processing

##### **GET**

Receive one record

##### **PUT**

Send one record

##### **CLOSE**

Terminate processing

##### **NCLOSE**

Abnormally terminate processing

##### **READ**

Receive one NJE record

##### **WRITE**

Send one NJE record

#### **DCT=**

Specifies either a pointer to a DCT or the address of a DCT that represents the remote terminal device; if a read or write, it represents a line DCT. If dct is written as an address, it represents the address of a fullword, which in its three rightmost bytes contains the address of the remote terminal device DCT. This word must be located on a word boundary in storage. If dct is written using register notation (either regular or special register notation), it represents the address of the remote terminal device DCT.

#### **PARM=**

If type specifies either OPEN, CLOSE, NCLOSE, READ, or WRITE, this parameter should not be specified. If type specifies GET, this parameter specifies the address of an area into which the input record will be placed. The input area must be defined large enough to contain the largest record to be received.

If the type-code is specified as PUT, this keyword specifies the address of a parameter area containing a CCW command code which contains the carriage control (or stacker select), the data length, and the starting address of the data in the following format:

##### **AL1**

CCW command word



## Registers on entry

**R0 - R15:**  
N/A

## Registers on exit

**R0 - R13:**  
Unchanged

**R14**  
Destroyed

**R15:**  
Return code

## Return codes

The following return codes (in decimal) are returned in register 15.

Return Code	Meaning
-------------	---------

**0**

Queue was empty before blocking.

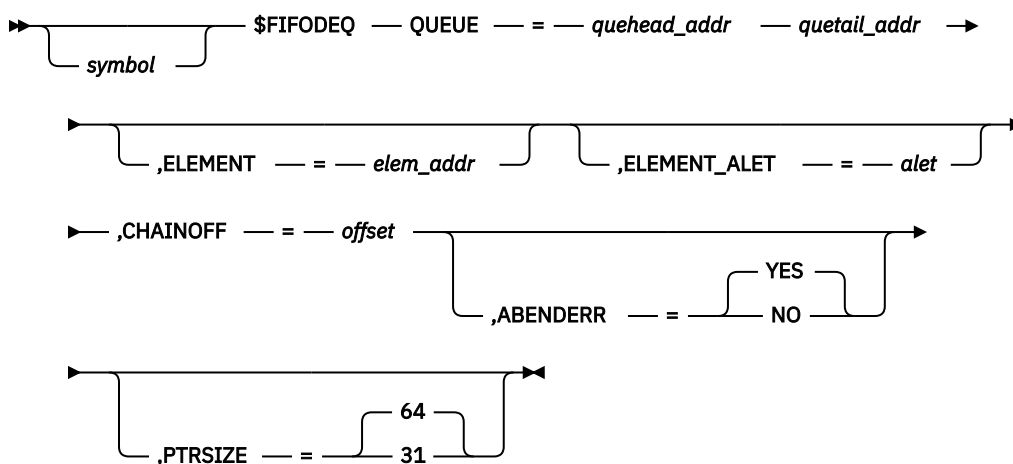
**4**

Queue was not empty, and there is no change made.

## \$FIFODEQ – Remove an element from a FIFO queue

Use the \$FIFODEQ macro to remove an element from a FIFO queue. The caller gives the address of a queue head, a queue tail, and the offset of a next and previous word for the work element and the address of the work element itself.

## Format description



### QUEUE=

Specifies the address of the queue head and tail. The head and tail of the queue must reside in consecutive fullword fields. This parameter is required.

### ELEMENT=

Specifies the address of the element to be removed from the queue. If the address is not specified, the first element is removed. ELEMENT and ELEMENT\_ALET are mutually exclusive.

Specifies the ALET of the element to be removed. ELEMENT and ELEMENT\_ALET are mutually exclusive.

## CHAINOFF=

Specifies the offset of the chaining fields within the element. The forward and the backward pointers must reside in consecutive fullword fields within each element. This parameter is required.

## ABENDR=

If ABENDERR=YES, the service will ABEND if an attempt is made to remove an element from the queue that is determined to be not on the queue. If ABENDERR=NO, a return code of 4 is returned instead. The default is ABENDERR=YES.

**PTRSIZE=**

Size of the chaining pointers (head, tail, next, and previous) and the count field. Valid values are:

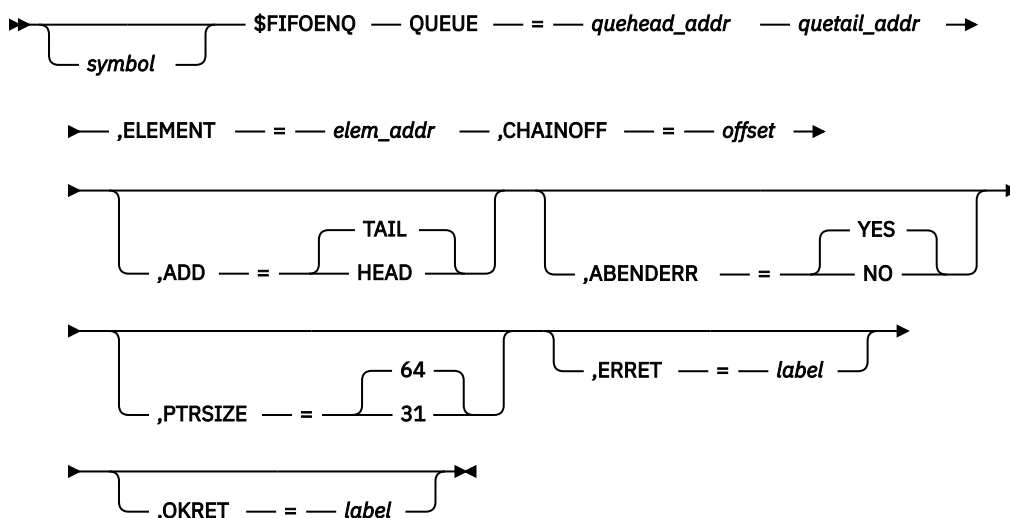
- 31 - Queue uses fullword pointers.  
64 - Queue uses doubleword pointers.

## Environment

- Any.

Use the \$FIFOENQ macro to queue an element onto a FIFO queue. The caller gives the address of a queue head, a queue tail, and the offset of a next and previous word for the work element and the address of the work element itself.

## Format description



**QUEUE=**

Specifies the address of the queue head and tail. The head and tail of the queue must reside in consecutive fullword fields. This parameter is required.

**ELEMENT=**

Specifies the address of the element to be added to the queue. This parameter is required.

## CHAINOFF=

Specifies the offset of the chaining fields within the element. The forward and backward pointers must reside in consecutive fullword fields within each element. This parameter is required.

## \$FIFOGTQ

### ADD=

Specifies how the element is to be added. ADD=TAIL specifies that the element is to be added at the end of the queue; ADD=HEAD indicates that the element is to be added at the front of the queue. The default is ADD=TAIL.

### ABENDERR=

If ABENDERR=YES, the service will ABEND if an attempt is made to remove an element from the queue that is determined to be not on the queue. If ABENDERR=NO, a return code of 4 is returned instead. The default is ABENDERR=YES.

### PTRSIZE=

Size of the chaining pointers (head, tail, next, and previous) and the count field. Valid values are:

31 - Queue uses fullword pointers.

64 - Queue uses doubleword pointers.

If PTRSIZE=64, the \$FIFOENQ macro must be called in 64-bit addressing mode. The default is PTRSIZE=31.

### ERRET=

Specifies label to receive control if an error is encountered (when ABENDERR=YES).

### OKRET=

Specifies label to receive control if no error is encountered (when ABENDERR=YES).

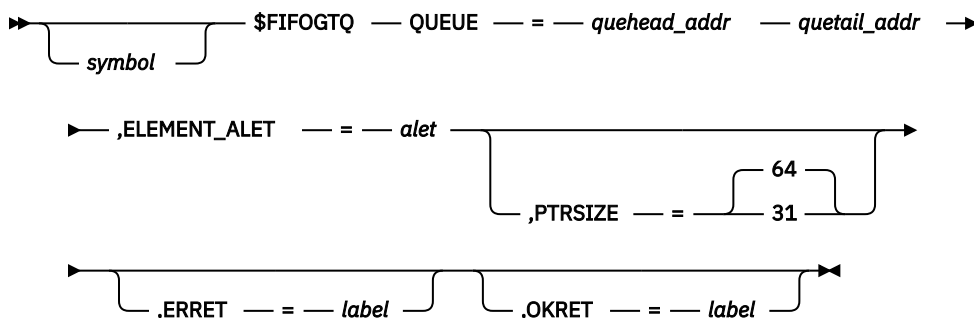
## Environment

- Any.

## \$FIFOGTQ - Dequeue an entire FIFO queue

Use the \$FIFOGTQ macro to de-chain an entire FIFO queue. The caller gives the address of a queue head and tail.

## Format description



### QUEUE=

Specifies the address of the queue head and tail. The head and tail of the queue must reside in consecutive fullword fields. This parameter is required.

### ELEMENT\_ALET=

Specifies the ALET of the element to be removed.

### PTRSIZE=

Size of the chaining pointers (head, tail, next, and previous) and the count field. Valid values are:

31 - Queue uses fullword pointers.

64 - Queue uses doubleword pointers.

If PTRSIZE=64, the \$FIFOGTQ macro must be called in 64-bit addressing mode. The default is PTRSIZE=31.



**ERRET=**

Specifies the label to receive control if the queue is empty. This parameter is optional.

**OKRET=**

Specifies the label to receive control if the queue is not empty. This parameter is optional.

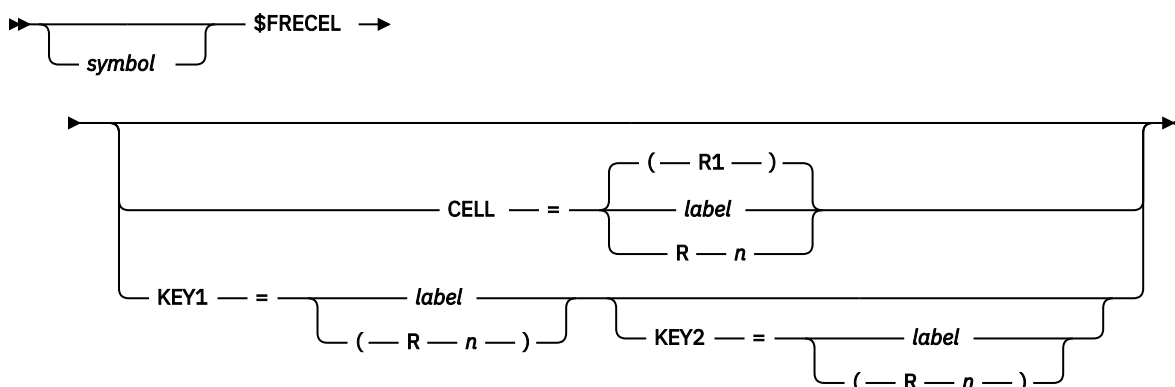
## Environment

- Any.

## \$FRECEL – Free an extended common storage area (ECSA) cell

Use \$FRECEL to return a single or multiple storage areas to the cell pool.

## Format description

**CELL=**

Specifies a label, or a register that contains the address, of the ECSA storage cell (previously obtained by a \$GETCEL) to be freed. If register notation is used, the designated register must be loaded with the address of the storage cell before this macro is issued. If this operand is omitted, CELL=(R1) is assumed.

**Note:**

1. The CELL= keyword is used to free a single cell.
2. The storage cell to be freed must have, as its first word, the address of the CCE associated with the storage area. The proper form for this address is obtained by the execution of a \$GETCEL macro instruction and should remain unaltered.
3. Either use the CELL= keyword or KEY1= and/or KEY2=. The CELL= and KEYn= keywords are mutually exclusive.

**KEY1=**

Specifies a label, or register that contains the address, of a key value (as set by the KEY1= keyword on the \$GETCEL macro) of the ECSA storage cells that are to be freed. All cells with the same KEY1= value will be freed. If you specify KEY1=, KEY2= is not required.

**KEY2=**

Specifies a label, or register that contains the address, of a key value (as set by the KEY2= keyword on the \$GETCEL macro) of multiple ECSA storage cells that are to be freed. KEY2= further specifies that a second key match is required to free all ECSA cells matching both the KEY1= and KEY2= values. Do not specify KEY2= unless you also specify KEY1=; however, KEY2= is not required.

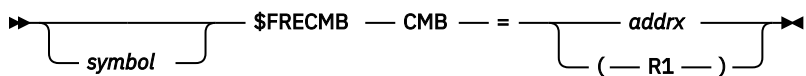
## Environment

- JES2 main task, JES2 subtask, FSS, or user environment.
- \$WAIT cannot occur.

## \$FRECMB – Free a console message buffer

Use \$FRECMB to return a console message buffer to the free queue.

### Format description



#### CMB=

Specifies the address of the console message buffer to be placed on the free queue. If register notation is used, the address of the console message buffer must be loaded into the designated register before executing this macro instruction.

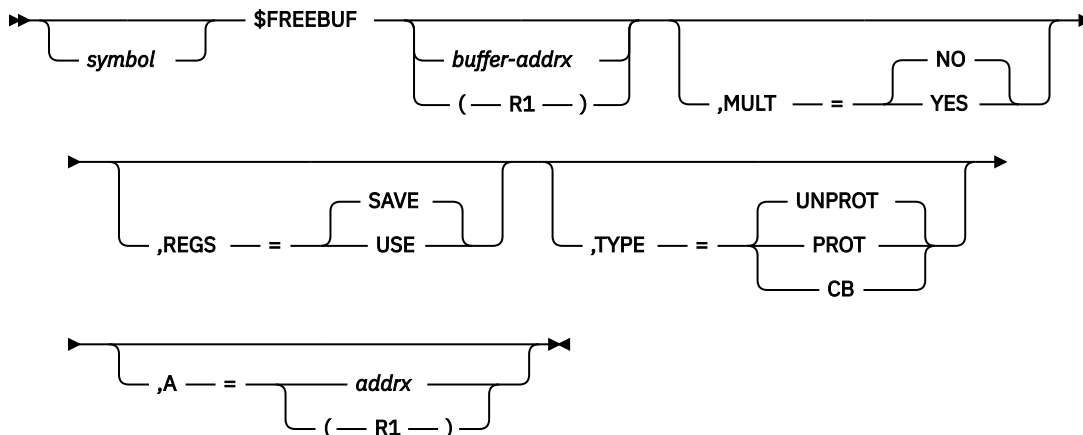
### Environment

- Main task.
- \$WAIT cannot occur.

## \$FREEBUF – Return a JES2 buffer to the JES2 buffer pool

Use \$FREEBUF from the JES2 main task, user, or FSS environments to return a JES2 buffer to the JES2 buffer pool.

### Format description



#### buffer

Specifies either a pointer to a buffer or the address of a buffer to be returned to the buffer pool as follows:

- If buffer is written as an address, then it represents the address of a full word which contains the address of the buffer to be returned in its three rightmost bytes.
- If buffer is written using register notation (either regular or special register notation), then it represents a register containing the address of the buffer to be returned.
- If register notation is used, the address must have been loaded into the designated register before the execution of this macro instruction.

**Note:** If you code TYPE=, you must specify the address of the buffer as the value of A=.

#### MULT=

Indicates that the specified buffer is the first of a chain of buffers, linked through their BUFCHAIN fields. If MULT=YES is specified, the entire chain is returned to the buffer pool. If the parameter

is omitted, only the specified buffer is returned to the pool. MULT=YES is not supported when TYPE=UNPROT is specified.

**REGS=**

Specifies whether (SAVE) or not (USE) to save contents of the caller's registers passed to the \$FREEBUF routine. REGS=SAVE is the default.

**TYPE=**

This parameter is only valid from the user environment. It returns the buffer from a specific subpool.

**PROT**

Return a protected user I/O buffer.

**UNPROT**

Return an unprotected user I/O buffer.

**CB**

Return a user control block buffer.

## SUBST

Return an in-stream symbol substitution buffer.

**A=**

Specifies the starting address of the buffer to be returned. This address is loaded into register 1.



**Attention:**

The specified buffers must have been obtained by a \$GETBUF macro instruction. Otherwise, the action of the macro instruction is unpredictable.

TYPE= and A= apply only to the user environment and should be omitted in the JES2 main task environment.

## Environment

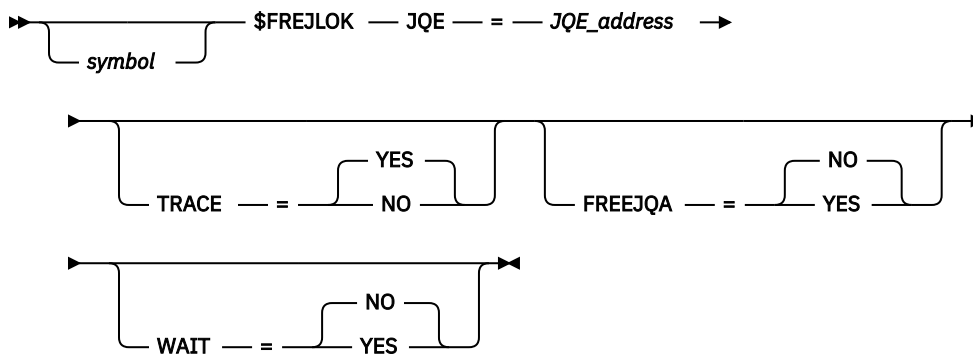
- Main task, subtask, user, or FSS environment.
- \$WAIT cannot occur.

## \$FREJLOK – Release the JES2 job lock

Use \$FREJLOK to release the JES2 job lock for the specified JQE address.

**Note:** Document in your code the reason for executing the \$FREJLOK macro instruction.

## Format description



**JQE=JQE\_address**

**Required.** Specifies the address of a fullword containing the address of the specified JOE to be released in its three right-most bytes. JOE= must be specified for TYPE=JOB.

## **\$FREJLOK**

### **TRACE=YES|NO**

Optional. Specifies whether to \$WAIT for the JOB lock to be obtained. This keyword only applies to a TYPE=JOB request; otherwise, it is ignored. The default value is YES.

### **RETJQA=YES|NO**

Optional. Specifies whether the JQA passed in R1 is returned via the \$DOGJQE macro.

### **WAIT=YES|NO**

Optional. WAIT=NO automatically defers the unlocking of the JQE being automatically deferred if either of the following conditions are met:

1. A real JQE or a READ mode JQA is provided by the JQE= parameter and the BERT lock is not available.
2. An update mode JQA is provided and it takes more BERTs to hold the data for the job than it did when the UPDATE mode JQA was obtained, and there are not enough BERTs to satisfy the request.

The deferred request is handled by DILBERT. The caller is not notified that the unlocking has been deferred.

The following register contents apply when \$FREJLOK is invoked:

#### **Return Code**

##### **Meaning**

#### **0-10**

Not applicable.

#### **11**

HCT address.

#### **12**

Not applicable.

#### **13**

PCE address.

#### **14-15**

Not applicable.

The following register contents apply on exit from \$FREJLOK:

#### **Return Code**

##### **Meaning**

#### **0-13**

Unchanged.

#### **14**

Used for linkage.

#### **15**

Return code:

#### **0**

The JES2 job lock has been released.

#### **8**

The JES2 job was already released.

#### **12**

The JES2 job lock has not been released because the job is being moved.

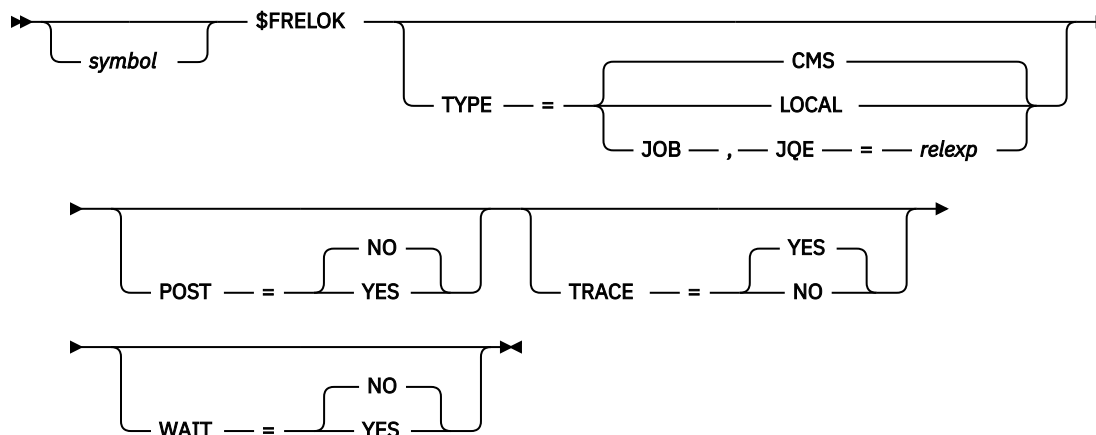
## **Environment**

Only the main task environment can use this macro.

## \$FRELOK – Free the MVS CMS lock, LOCAL, or JES2 job lock

Use \$FRELOK to free the CMS lock, LOCAL, or JES2 job lock obtained through the \$GETLOK macro instruction, and specify any JES2 follow-up processing.

### Format description



#### TYPE=

Specifies the lock to be freed as follows:

##### **CMS (default)**

The cross-memory services lock is freed. All other operands are ignored.

##### **LOCAL (valid in the HASPFSSM environment only)**

The local lock is freed. All other operands are ignored.

##### **JOB (valid in the JES2 main task only)**

The JES2 job lock is freed; you *must* specify a job queue element address (JQE=).

#### JQE=

Specifies the address of a fullword containing the address of the JQE.

#### POST=

Specifies whether (YES) or not (NO), JES2 should issue a \$#POST when the indicated lock is freed.

#### **Note:**

1. POST= is only valid if you code TYPE=JOB.
2. If you specify POST=NO, JES2 does not issue a \$#POST, possibly causing SYSOUT to not be selected by ready devices. Therefore, if you do specify POST=NO, do so only if:
  - No JOEs or null data sets were created for the job or
  - Only spin data sets exist for the job.

#### TRACE=

Specifies whether (YES) or not (NO) a rolling trace entry should be created when the indicated lock is freed.

#### WAIT=

WAIT=NO (NO is the default) results in the unlocking of the JQE being automatically deferred if:

- A real JQE or a READ mode JQA is provided through the JQE= parameter and the BERT lock is not available.
- An UPDATE mode JQA is provided and it takes more BERTs to hold the data for the job than it would when the UPDATE mode JQA is obtained and there aren't enough BERTs to satisfy the request.

The deferred request is handled through \$DILBERT. The caller is not notified that the unlocking has been deferred.

## Environment

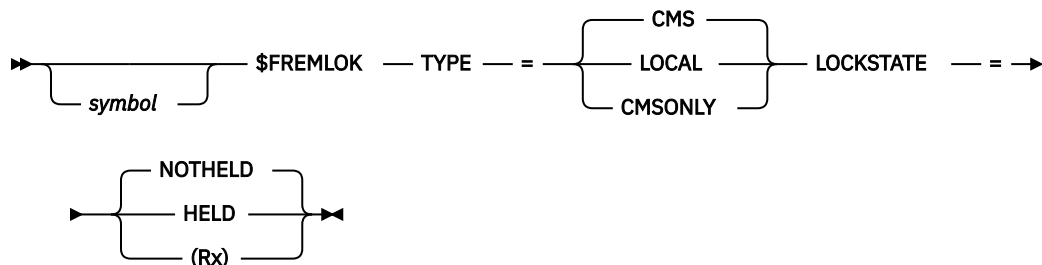
- Main task and functional subsystem (HASPFSSM).
- \$WAIT can occur.

## \$FREMLOK – Release the MVS CMS or LOCAL job lock

Use the \$FREMLOK macro to release the CMS or LOCAL lock, or both.

**Note:** Document in your code the reason for executing the \$FREMLOK macro instruction.

## Format description



### TYPE

Specifies the type of lock to manage. The following values are supported:

#### LOCAL

Releases the MVS LOCAL lock only.

#### CMS

Releases the CMS lock and the MVS LOCAL lock.

#### CMSONLY

Releases the CMS lock only.

### LOCKSTATE

Specifies whether the LOCAL lock was held when the corresponding \$GETMLOCK was invoked.

LOCKSTATE is only needed when TYPE=CMSONLY is specified. The following values are supported:

#### HELD

The LOCK was already held.

#### NOTHELD

The default. The local lock was not held.

#### (Rx)

A register with a 0 (lock not held) or NZ value (lock was already held).

The following register contents apply when \$FREMLOK is invoked:

### Return Code

#### Meaning

#### 0-10

Not applicable.

#### 11

HCT/HCCT/HFCT address, as applicable.

#### 12

Not applicable.

#### 13

PCE/Save area address, as applicable.

#### 14-15

Not applicable.

The following register contents apply on exit from \$FREMLOK:

**Return Code****Meaning****0-13**

Unchanged.

**14**

Used for linkage.

**15**

0

## Environment

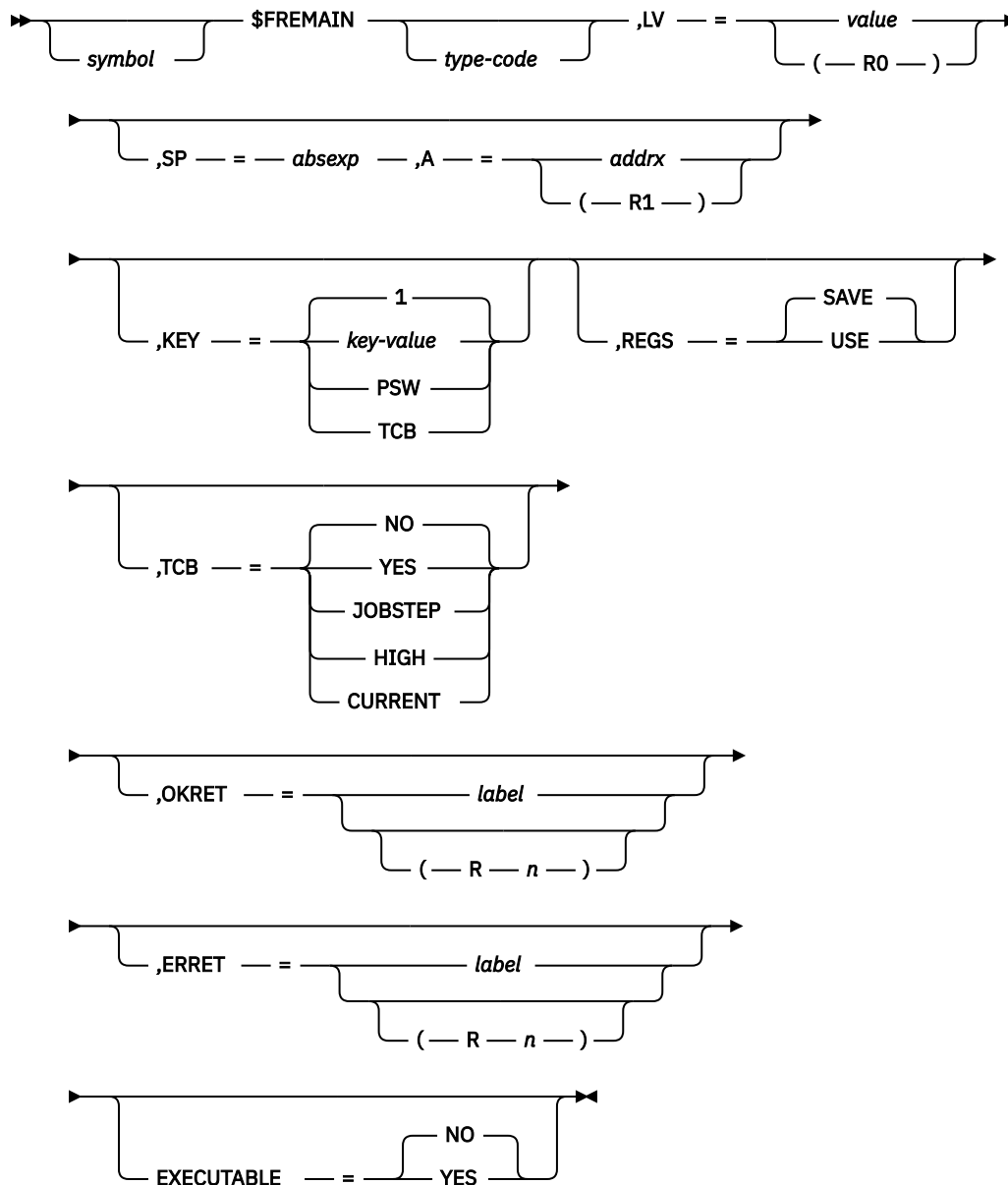
The Main, User, Subtask and FSSM environments can use this macro.

## **\$FREMAIN – Branch-entry FREEMAIN services**

---

Use \$FREMAIN to free an area of storage that is obtained through MVS GETMAIN or \$GETMAIN services.

## Format description



### type-code

Identifies the type of GETMAIN/FREEMAIN request. You can only specify an unconditional FREEMAIN request.

#### Note:

If the area of storage referred to has not been obtained through MVS GETMAIN or \$GETMAIN services before this point, your subsystem abends.

The way to specify requests is as follows:

R or RU or U – An unconditional FREEMAIN request.

### LV=

Specifies the length of the area to be obtained or freed. This value is loaded into register 0. When this value is coded by way of register format, the subpool can be specified in the high order byte of the register.



**SP=**

Identifies the subpool number. Subpool zero is the default if no subpool is specified or the subpool is not specified in the high order byte of the LV= parameter. This parameter must be specified if you want to free an entire subpool. (In that case, do *not* code the A= parameter.)

**A=**

Specifies the starting address of the storage to be freed. This is done either by loading the address into register 1 and coding R1 as the value of A=, or by specifying the address itself. This address is loaded into register 1. This parameter is required unless you need to free an entire subpool, in which case you specify the SP= parameter and not the A= parameter. This keyword applies to the JES2 main task environment.

For the user environment, specifies the address of the TCB associated with the storage to be freed; this parameter is required for this environment.

**KEY=**

Specifies the key of the storage you want to free. The default is "1". Specifying "TCB" indicates to use the current TCB. This keyword applies to the user environment. Specifying "PSW" is valid for only the user environment.

**REGS=**

Specifies whether (SAVE) or not (USE) to save the contents of the caller's registers. This keyword is valid only in the user environment. REGS=SAVE is the default.

**TCB=**

Indicates which TCB is associated with the storage that is to be freed.

**YES**

Specifies that the TCB to be used has its address in the first word of the storage to be freed.

**NO**

Specifies that the TCB to be used is the jobstep TCB. If there is no jobstep TCB, then the current TCB is used. This is the default.

**JOBSTEP**

Specifies that the jobstep TCB is to be used.

**HIGH**

Specifies that the highest TCB in the address space (whose address is in ASXBFTCB) is to be used.

**CURRENT**

Specifies that the current TCB is to be used.

**OKRET=**

Specifies a label or a register containing the address of the routine that receives control if \$FREMAIN processing was successful.

**ERRET=**

Specifies a label or a register containing the address of the routine that receives control if an error occurs during \$FREMAIN processing.

**EXECUTABLE=**

Specifies the type of storage that is freed. The EXECUTABLE= parameter on the \$FREMAIN call must match the \$GETMAIN call. The default is EXECUTABLE=NO. If there is a mismatch in the EXECUTABLE= parameter on the \$FREMAIN call, a program check occurs.

**NO**

The storage that is obtained is non-executable.

**YES**

The storage that is obtained is executable.

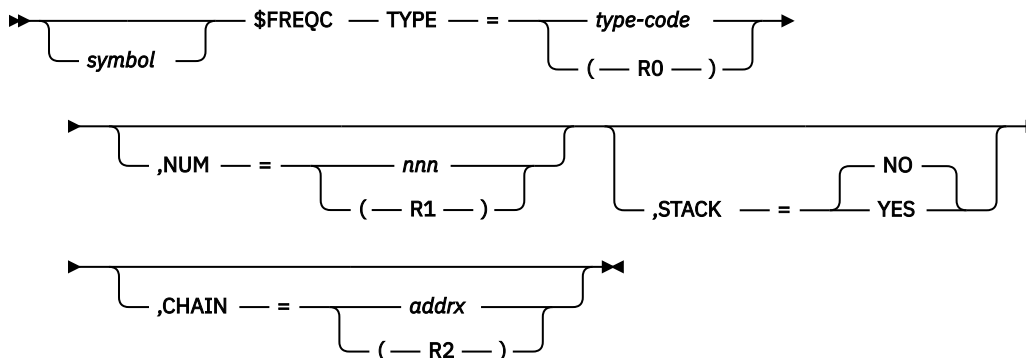
## Environment

- Main task or user address space.
- \$WAIT cannot occur.

## \$FREQC – Free quick cell

Use \$FREQC to free the storage obtained for the quick cells.

### Format description



#### TYPE=

Specifies the type of quick cells that are to be freed.

##### type-code

Specifies the type code as defined in the \$QCTGEN macro for the quick cell to be freed.

##### (R0)

Specifies that register 0 contains the quick cell type-code as defined in the \$QCTGEN macro. The two low-order bytes of the register must contain the type-code and the two high-order bytes must be zeroed.

**Note:** This keyword must be specified or an error will occur at assembly time.

#### NUM=

Specifies the number of quick cells to be returned to the quick cell pool.

##### nnn

Specifies the number (1-255) of quick cells. The value assigned to NUM= must not exceed the specification of QCTLIMIT; exceeding this value causes an execution error (F01 FSI catastrophic error), and the \$HASP750 message is issued. The default is 1.

##### (Rn)

Specifies the register where the number of quick cells is held.

#### STACK=

Specifies whether the quick cells should be dechained or chained together. The chaining field offset is specified in the QCT.

##### YES

Specifies that the quick cells specified by this macro are taken off a stack identified by the QCT for the specified type-code.

##### NO

Specifies that the quick cells specified by this macro are chained together and can be freed one at a time while progressing through the chain.

#### CHAIN=

Specifies the register that contains the address of the first cell of the chain of quick cells that are to be freed. Register 0 and 1 cannot be used. This keyword must be coded if STACK=NO is coded or allowed to default.

## Environment

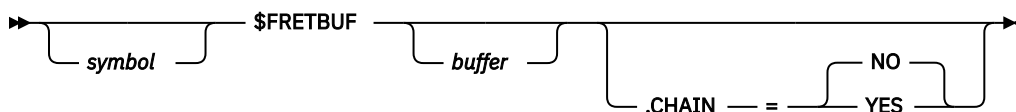
- Functional subsystem (HASPFSM).

- \$WAIT not applicable.

## \$FRETBUF – Free TCP buffer

Use \$FRETBUF to free a TCP buffer or chain of TCP buffers.

### Format description



#### buffer

Address of the first TCP buffer to be freed

#### CHAIN=

Specifies whether the specified buffer or a chain of buffers should be freed.

#### YES

Indicates that the specified buffer is to be freed.

#### NO

Indicates that a chain of buffers is to be freed.

### Environment

JES2 address space or NETSRV address space.

### Return codes

#### Return Code

#### Meaning

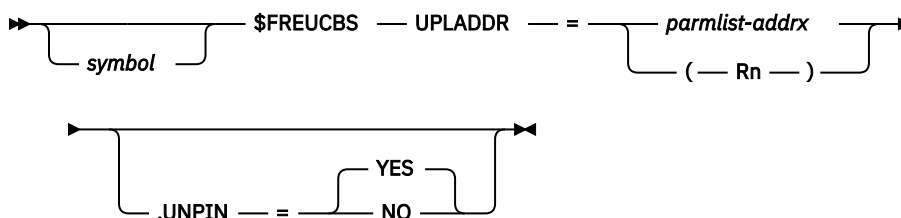
0

Buffer freed

## \$FREUCBS – Free UCB parameter list storage

Use \$FREUCBS to free UCB parameter list (UPL) storage and, optionally, to unpin the corresponding UCB.

### Format description



#### UPLADDR=

Specifies the address of the UCB parameter list that is to be freed. If register notation is used, the register must be initialized with the UCB parameter list address before the execution of the macro.

#### UNPIN=YES|NO

Specifies whether the system is (YES) or is not (NO) to unpin the UCB identified in the UPL. UNPIN= is optional with a default of YES.

## Environment

- Main task.
- \$WAIT cannot occur.

## \$FREUNIT – Release a unit device control table (DCT)

Use \$FREUNIT to release a device control table (DCT).

### Format description



#### dct

Specifies either a pointer to a DCT or the address of a DCT to be released. If *dct* is written as an address, then it represents the address of a full word, which in its three rightmost bytes contains the address of the DCT to be released. If *dct* is written using register notation (either regular or special register notation), then it represents the address of the DCT to be released. If register notation is used, the address must be loaded into the designated register before the execution of this macro instruction.

#### Note:

1. The execution of this macro instruction may cause a \$WTO macro instruction to be executed.
2. When a device that was allocated by MVS allocation facilities goes into the DRAINED status, the device is deallocated. To use the device, it must first be started by the operator using the \$S device command and the device must be obtained through the \$GETUNIT macro instruction. Otherwise, the system replies device unavailable.

#### Note:

The specified DCT must have been obtained by a \$GETUNIT macro instruction. The action of the macro instruction is unpredictable in other cases.

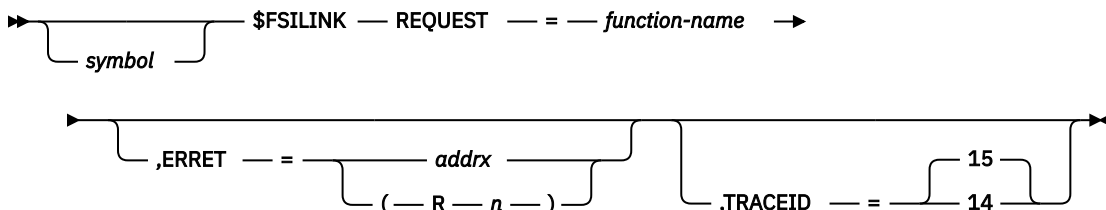
## Environment

- Main task.
- \$WAIT can occur.

## \$FSILINK – Link the functional subsystem interface

Use \$FSILINK to provide base register setup for the major control blocks required by the JES2 functional subsystem interface (FSI) service routines. Specify this macro at the entry point of an FSI service routine. \$FSILINK sets up registers to point to the functional subsystem control block and functional subsystem application control block.

### Format description



**symbol**

A symbol **must** be specified on this macro instruction.

**REQUEST=**

Specifies the function id of this FSI service. The function id specified is compared to that passed in the FSI parameter list at the time of the FSI call. If they do not match, a return code of 4 is placed in register 15 and, if specified by the ERRET= keyword, a branch is taken to an error routine.

**ERRET=**

Specifies the label or a register containing the address to branch to if an invalid function id was specified on the REQUEST= keyword.

**TRACEID=**

Specifies whether trace id 14 (trace GETDS, RELDS, SEND) or trace id 15 (trace GETREC, FREEREC, CHKPT) is to be used for tracing. This macro supports these two trace ids only.

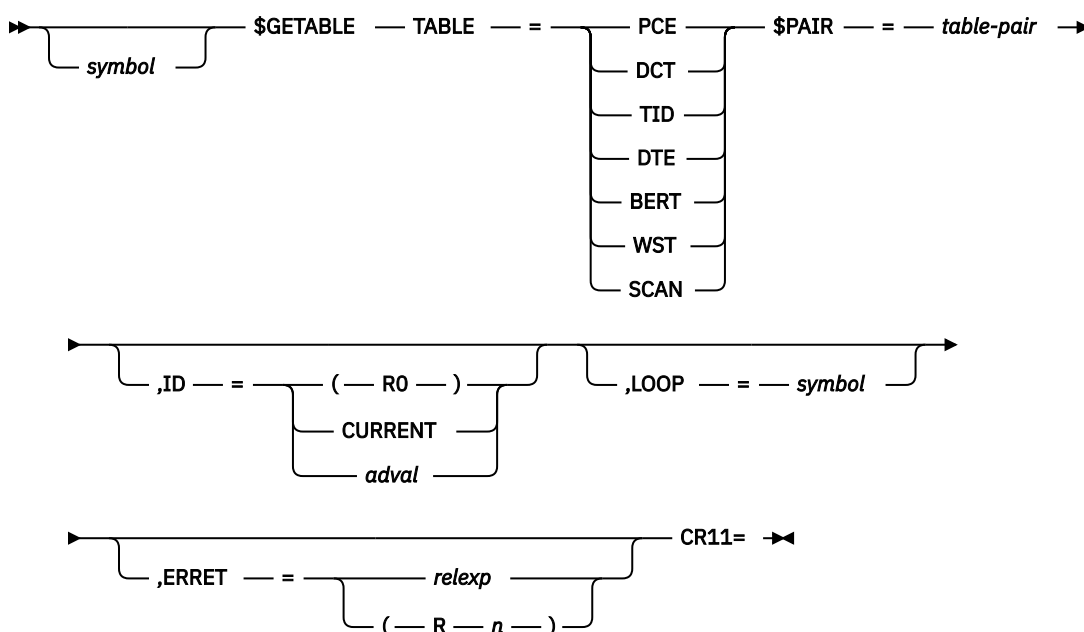
## Environment

- Functional subsystem (HASPFSM).
- MVS WAIT can occur.

## \$GETABLE – Get HASP/user table entries

Use the \$GETABLE macro to return table entries of the HASP/USER table pairs.

## Format description

**TABLE=**

Specifies the type of table pairs to be used.

**PCE**

Indicates the PCE table (\$PCETAB).

**TID**

Indicates the trace id table (\$TIDTAB).

**DCT**

Indicates the DCT table (\$DCTTAB).

**DTE**

Indicates the DTE table (\$DTETAB).

## **\$GETADDR**

### **BERT**

Indicates the BERT table (\$BERTTAB).

### **WST**

Indicates the work selection table (\$WSTAB).

### **SCAN**

Indicates the \$SCAN tables (\$SCANTAB).

### **\$PAIR=**

Specifies the name of the table pair to use to search for tables. This operand is required when TABLE=WST or TABLE=SCAN.

### **ID=**

Specifies the table entry id associated with the table entries to be returned to the table pairs.

CURRENT indicates that the id for the current environment is to be used – that is, PCEID (for TABLE=PCE) or TTEID (for TABLE=TID), DCTDEVID (for TABLE=DCT), or DTESTID (for TABLE=DTE). If register notation is used, the designated register must contain the table entry id before executing this macro.

### **LOOP=**

Specifies a label that serves as the terminating point of the loop that is the table entries.

If LOOP= is omitted, a single table entry lookup is performed.

### **ERRET=**

Specifies the address of the error routine that is to receive control if the table entry is invalid or the end of the tables is reached.

### **Note:**

1. ID= and/or LOOP= must be specified.
2. You must preserve general purpose register 0 and access register 0 before executing \$GETABLE in a loop.

## **Environment**

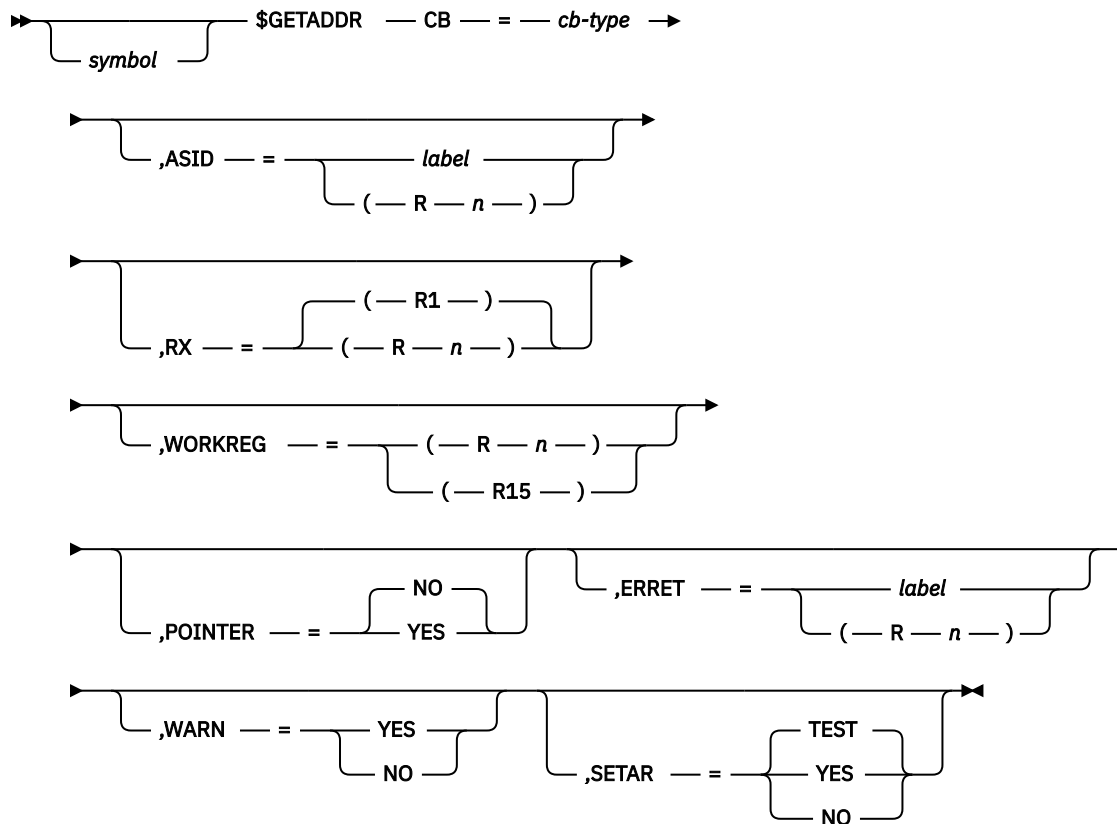
- USER, SUBTASK, or FSS might be allowed, if TABLE=SCAN.
- Main task and during JES2 initialization and termination.
- \$WAIT cannot occur.

## **\$GETADDR – Get a control block address**

---

Use the \$GETADDR macro instruction to obtain the address of a specific control block.

## Format description



### CB=

Specifies the control block type (cb-type) as follows:

#### Common storage control blocks:

##### ASCB

Address space control block

##### ASSB

Address space secondary block

##### CADDR

Common address routine table

##### FSSCB

Functional subsystem control block

##### HASB

HASP address space block

##### HAVT

HASP address vector table

##### HCCT

HASP common communications table

##### PCT

Path Manager Control Table

##### UCADDR

User's common address routine table (CSA)

#### Private storage control blocks:

##### ASXB

Address space extension block

**FSCT(JES)**

Functional subsystem control table

**FSVT**

Functional subsystem vector table

**HASXB**

HASP address space extension block

**HCT**

HASP control table

**HFCT**

HASP functional subsystem communication table

**PADDR**

Private address routine table

**SSIB**

Subsystem information block

**SSICLLR**

Subsystem interface caller's save area

**SSOB**

Subsystem options block

**TRE**

TCB recovery element

**UPADDR**

User's private address routine table

**Note:** Not all control blocks are accessible from all address spaces. \$GETADDR issues a warning message (if WARN=YES is specified) if you request a control block that cannot be accessed.

**ASID=**

Specifies a half-word field, or a register that contains, the address space ID (ASID) where the control block resides. If this ASID is different from the current address space, you can request only common storage area-resident (CSA) control blocks. ASID= defaults to the current ASID.

**RX=**

Specifies the register which is to contain the requested control block address following \$GETADDR processing. **Register 0 should not be used.**

**WORKREG=**

Specifies a register (R1-R15) that \$GETADDR can use for work. You can specify a null value (WORKREG=); however, if \$GETADDR requires a work register, you will receive assembly errors. This keyword is optional; register 15 is used if a register is needed. On return to the caller, the contents of register 15 (or the specified register) will be unpredictable. **Register 0 should not be used.**

**POINTER=**

YES indicates that the address of a 4-byte pointer to the control block is to be returned to the caller in the register specified by RX=.

NO will cause the address of the control block to be returned to the caller in the register specified by RX=.

**ERRET=**

Specifies a label or register that contains the address of the routine that receives control if this macro processing was unable to obtain the requested control block address. The ERRET= specification does not apply to ASCB, ASXB, CADDR, HASB, HASXB, HAVT, HCCT, and UCADDR control block types. If you do not specify ERRET= and \$GETADDR is unsuccessful the register specified by RX= will contain a zero on return.

**WARN=**

Specifies whether (YES) or not (NO) JES2 will issue messages noting that a particular control block cannot be accessed from the current environment. WARN=YES is the default.



**SETAR=YES|NO|TEST**

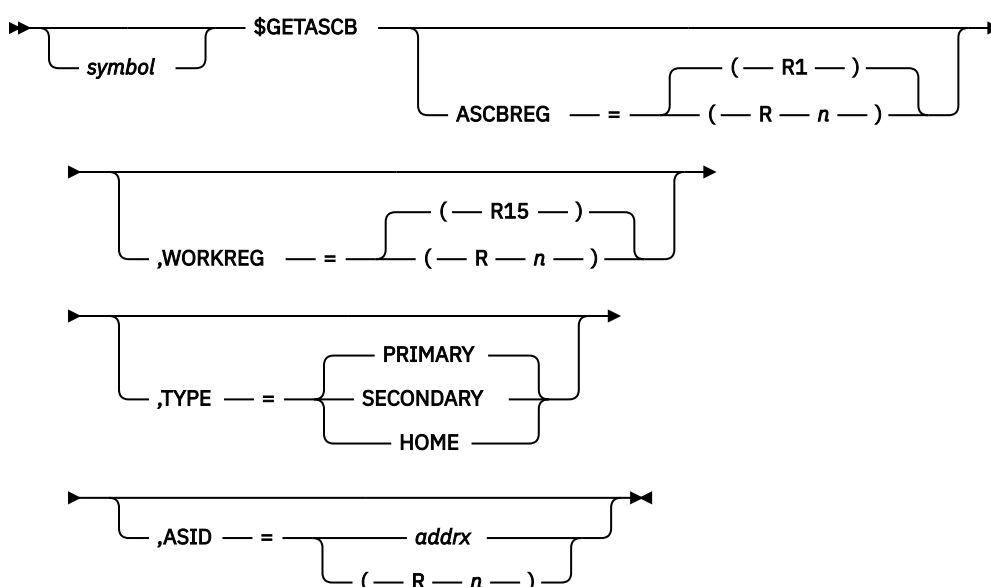
Specifies whether the access register associated with the RX parameter should be set as well as the general purpose register. The default is SETAR=TEST.

**Environment**

- All environments.
- \$WAIT can occur.
- Callers in AR ASC mode are supported.

**\$GETASCB – Retrieve the primary, secondary, or home ASCB**

Use the \$GETASCB macro instruction for a specific asid to place the primary, secondary, or home ASCB address into a register.

**Format description****ASCBREG=**

Specifies the register where the ASCB address is to be stored. Register 1 is the default.

**WORKREG=**

Specifies a work register that can be used in processing your request. Register 15 is the default.

**TYPE=**

Specifies the type of ASCB that is to be located.

**PRIMARY (the default)**

indicates the primary address space's ASCB.

**SECONDARY**

indicates the secondary address space's ASCB.

**HOME**

indicates the home address space's ASCB.

**ASID=**

Specifies the address of the storage location containing the ASID for the address space whose ASCB address is to be returned or specifies the register containing the ASID.

**Note:** The TYPE= operand is ignored if ASID= is specified.

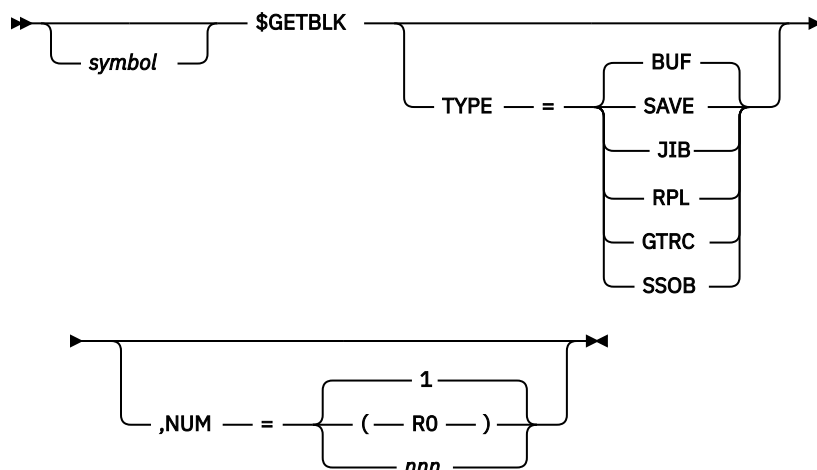
## Environment

- Main task, subtask, or user address space.
- \$WAIT cannot occur.

## \$GETBLK – Get a storage cell from a free cell pool

Use \$GETBLK to obtain a specified number of predefined storage cells from one of several free cell pools.

### Format description



#### TYPE=

Specifies the type of storage cell that is to be obtained and from which cell pool the cell is to be obtained. The following storage cell types may be coded:

##### Cell Type

##### Meaning

##### SAVE

An MVS-type save area

##### JIB

A JOE information block

##### BUF

An I/O buffer of 4K bytes

##### RPL

A request parameter list control block chain

##### GTRC

A GETREC chain control block

##### SSOB

A subsystem options block

#### NUM=

Specifies the number of storage cells that are to be obtained. Specify this number (*nnn*) as a valid number not greater than that specified on the \$QCTGEN macro LIMIT= keyword or place the number in a register (Rn). If STACK=YES was coded on the \$GETQC macro, the cells specified by this macro are chained in a stack. If NUM= is specified as greater than 1, the blocks will be chained using the chain field specified in the QCT for that storage type.

## Environment

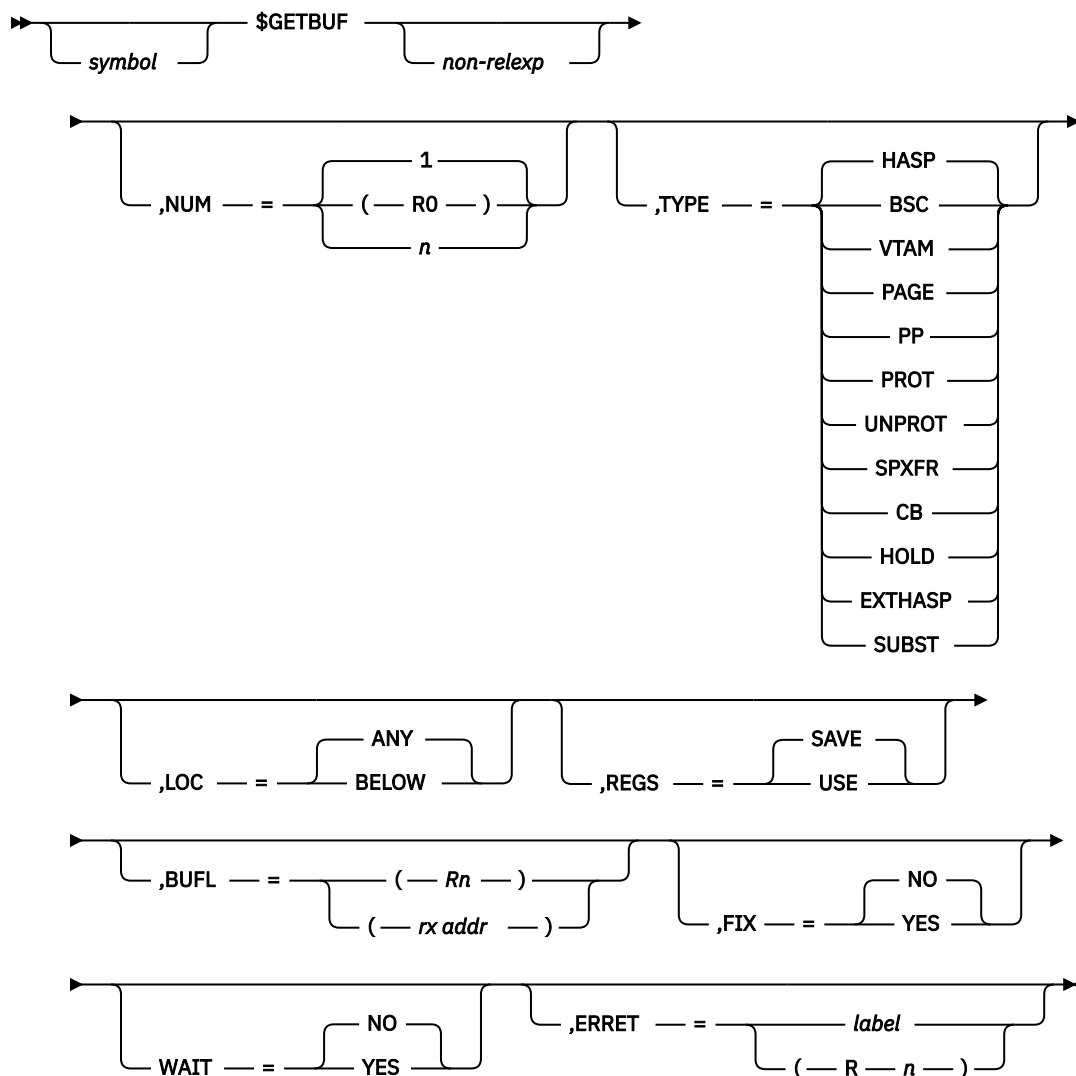
- Functional subsystem (HASPFSM).

- MVS WAIT can occur.

## \$GETBUF – Acquire a buffer from a JES2 buffer pool

Use \$GETBUF in the JES2 main task, user, or FSS environments to obtain a buffer from a buffer pool and return the address of this buffer in register 1.

### Format description



#### none

Specifies a location to which control is returned if no buffers are available. (If WAIT=YES is specified, this operand is ignored.)

If this operand is omitted, the condition code is set to reflect the availability of a buffer as follows:

#### CC=0

No buffer is available.

#### CC≠0

R1 contains the address of the buffer.

**Note:** This is only valid in the main task environment.

**NUM=**

Specifies the number of buffers or a register containing the number of buffers to be obtained. (This keyword is ignored if this macro instruction is issued from the user or subtask environment.) One buffer is the default.

**TYPE=**

Specifies the type of buffer to be obtained, and whether the buffer is to contain an IOB or an RPL, by type code as follows:

**HASP (default for main task)**

A local buffer where an input/output buffer (IOB) is to be constructed.

**Note:** This is only valid in the main task environment.

**BSC**

A teleprocessing (TP) buffer where an IOB is to be constructed.

**Note:** This is only valid in the main task environment.

**VTAM**

A TP buffer where a request parameter list (RPL) is to be constructed.

**Note:** This is only valid in the main task environment.

**PAGE**

A local 4096-byte buffer where an IOB is to be constructed.

**Note:** This is only valid in the main task environment.

**PP**

A local print/punch buffer where an IOB is to be constructed.

**Note:** This is only valid in the main task environment.

**PROT**

A protected buffer where an IOB is to be constructed.

**Note:** This is only valid in the user and subtask environments.

**UNPROT**

An unprotected buffer where an IOB is to be constructed.

**Note:** This is only valid in the user and subtask environments.

**SPXFR**

A local spool offload buffer.

**Note:**

1. This is only valid in the main task environment.
2. You must specify WAIT=YES if TYPE=SPXFR is specified.

**CB (default for user and subtask environments)**

A control block buffer is to be constructed.

**HOLD**

An unprotected buffer used for GET/UPDATE.

**Note:** This is only valid in the user environment.

**EXTHASP**

A HASP data buffer is to be constructed with the storage above the 16 megabyte line.

**SUBST**

In-stream symbol substitution buffer. This buffer type is only valid in user and subtask environments. This buffer type requires specification of buffer length: refer to the BUFL keyword.

**BUFL=**

Size of the buffer (in addition to the standard buffer header). This parameter is required for TYPE=SUBST. For other buffer types, buffer size is implicit and the BUFL keyword is ignored.

**BUFL=rx-addr**

Buffer size is in a word at the specified address.

**BUFL=(Rn)**

Buffer size in the specified register Rn.

**LOC=**

Specifies whether the buffer that is obtained can be above (ANY) or must be below (BELOW) 16 megabytes in virtual storage. LOC is valid only if TYPE= specifies PROT, UNPROT, HOLD, or CB. LOC=ANY is the default.

**REGS=**

Specifies whether (SAVE) or not (USE) to save the contents of the caller's registers. REGS= is valid only in the user or subtask environments. REGS=SAVE is the default.

**FIX=**

Specifies whether the buffer is to be page-fixed (YES); if FIX=NO is specified or this parameter is omitted, the page containing the buffer is not fixed.

**Note:** This is only valid in the main task environment.

**WAIT=**

Specifies whether the \$GETBUF service routine is to cause the routine issuing the macro to wait (\$WAIT BUF) until buffers are available (WAIT=YES), or whether control is to be returned immediately (WAIT=NO or parameter omitted) if buffers are not available. If TYPE=SPXR is specified, you must also specify WAIT=YES.

**Note:** This is only valid in the main task environment.

**ERRET=**

Specifies the label or a register (R2-R12) that contains the address of the routine that receives control if \$GETBUF does not successfully obtain the requested storage cells.

**Note:** TYPE=, REGS=, LOC=, and ERRET= are the only keywords applicable in the user environment.

**Attention:**

The JES2 buffer that is obtained by using the \$GETBUF macro contains a prefix area that must not be altered. This prefix area is used by the \$FREEBUF macro; unpredictable results may occur if the prefix area is altered.

## Return codes

The following return codes (in decimal) are returned in register 15.

**Return Code****Meaning**

**0**

Buffer obtained

**4**

Buffer not obtained

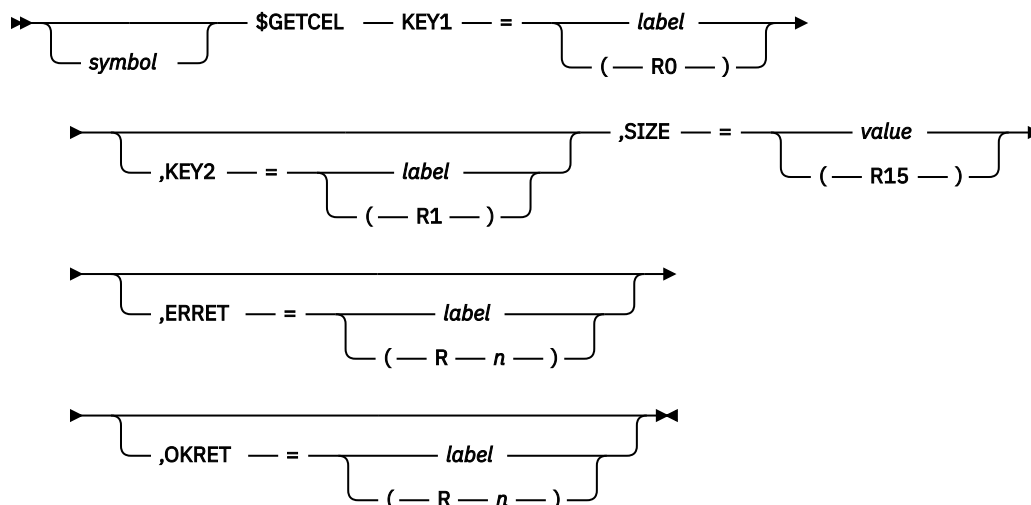
## Environment

- JES2 main task, subtask, or user environment.
- \$WAIT can occur if you specify WAIT=YES on the macro.

## **\$GETCEL – Acquire an extended common storage (ECSA) area cell**

Installation-written exit routines can use \$GETCEL to obtain storage area cell pools in ECSA for use in communicating between the JES2 main task and the user environment when user job and task ownership is required. To free a cell, use the \$FRECEL macro.

## Format description



### KEY1=

Specifies the label of, or a register that contains, a value used as the key value identifier for this storage cell. Only cells with the value specified for KEY1= will be freed unless KEY2= is specified, in which case storage cells that match both key values will be freed. This value can be used by the \$FRECEL macro when the storage cell is freed.

If register notation is used, the designated register must be loaded with the KEY1 value. This keyword is required.

### KEY2=

Specifies the label of, or a register that contains, a value used as the secondary key value identifier. This keyword is used to further identify the owner of the cell. Only storage cells that match both the value specified in KEY1= and KEY2= will be freed.

### SIZE=

Specifies the storage size in bytes of the cell. The value specified must be between 1 and 65280, inclusive. If register notation is used, the designated register must be loaded with the value before execution of this macro instruction.

**Note: If register notation is used, register 2 will be changed by the macro expansion.**

### ERRET=

Specifies the label or a register that contains the address of a routine that is to receive control if \$GETCEL is unable to obtain the requested storage cell. If ERRET= is specified and an error occurs, R15 will contain a return code of 4 on return to the caller.

### OKRET=

Specifies the label or a register that contains the address of a routine that is to receive control if JES2 can obtain the requested CSA storage cell.

### Note:

1. On obtaining a storage cell, the first word of the storage block contains the address of the controlling cell control element (CCE). This word must be left within the storage area in order for the \$FRECEL macro instruction to free the storage.
2. If SIZE= register notation is used, register 2 is changed.

## Return codes

The following return codes (in decimal) are returned in register 15.

Return Code	Meaning
-------------	---------

- 0  
storage cell obtained
- 4  
storage cell not obtained

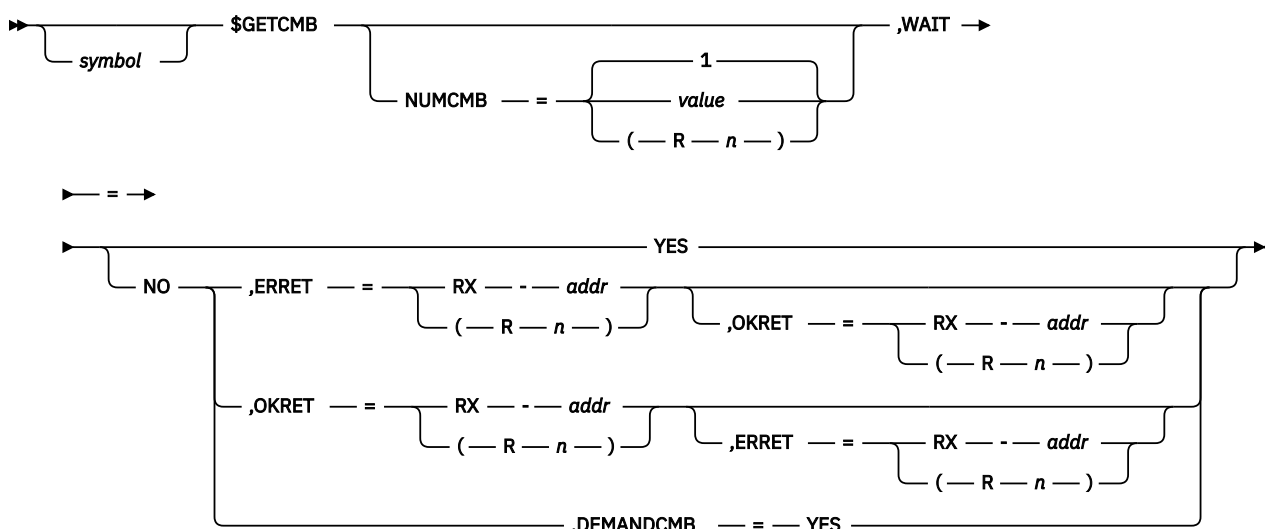
## Environment

- All environments.
- \$WAIT cannot occur.

## \$GETCMB – Get console message buffers

Use \$GETCMB to obtain one or more console message buffers from the free queue and return the address of the first buffer in register 1.

## Format description



### NUMCMB=

Specifies the number of console message buffers required. If register notation is used, the number of required console messages must be loaded into the designated register (R1 - R12) before the execution of this macro.

If this operand is omitted, NUMCMB=1 is assumed.

### WAIT=

Specifies the action to be taken in the event insufficient console message buffers (CMBs) are available to satisfy the request. This parameter **must** be coded.

### YES

\$GETCMB is to return control only after the request is satisfied. Register 15 contains a return code. If you code **YES**, do not code either **ERRET=** or **OKRET=**.

### NO

If the request cannot be satisfied, \$GETCMB is to return control immediately. Register 15 contains a return code. If you code **WAIT=NO**, you must also code **ERRET=**, or **OKRET=**, or both.

### ERRET=

Specifies the address of a routine that is to receive control if the request is unsuccessful. You can specify an RX-address or a register (R2-R12) that contains the address. There is no default.

**OKRET=**

Specifies the address of a routine that is to receive control if the request is successful. You can specify an RX-address or a register (R2-R12) that contains the address. There is no default.

**DEMANDCMB=YES**

Specifies that a console message buffer (CMB) is needed but the processor cannot wait. If you code this parameter, you must not code **ERRET=** or **OKRET=**.

## Return codes

The following return codes (in decimal) are returned in register 15:

Return Code	Meaning
0	The request was satisfied and register 1 points to the first CMB. If there are multiple CMBs, field CMBCMB is used to chain the CMBs. In the last CMB on the chain, CMBCMB contains zero.
4	The request could not be satisfied. Register 1 contains 0.

## Register contents when \$GETCMB returns control

Register	Contents
0	Unpredictable
1	Address of the CMB chain or 0.
2-13	Unchanged
14	Unpredictable
15	Return code.

## Environment

- Main task.
- \$WAIT can occur (if you specify WAIT=YES on the macro).

**Note:**

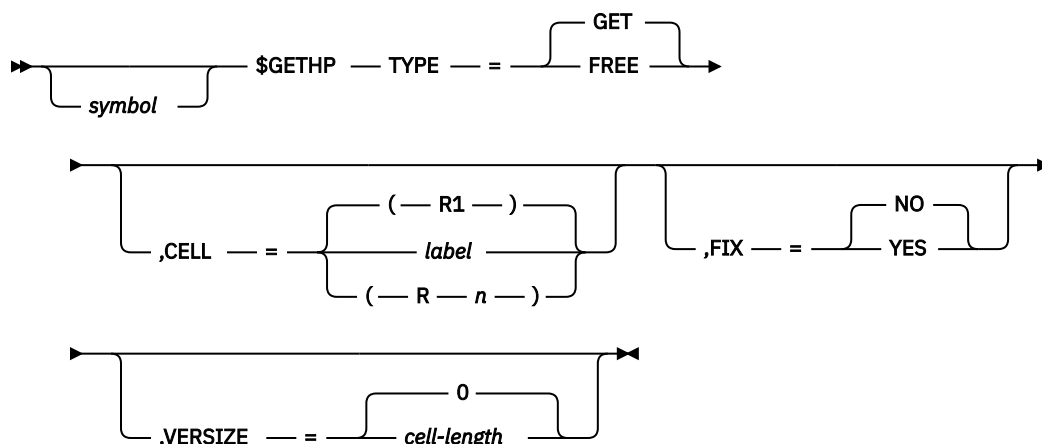
1. If you obtain CMBs and then \$WAIT (either directly or indirectly) on a resource owned by another processor (PCE) that issues a \$WTO, you might encounter a CMB lockout problem.
2. Do not use \$FRECMB to free a CMB that has been processed by a \$WTO that specified CMB=YES because the \$WTO has already freed the CMB.

## \$GETHP – Get high private cell pool

Use \$GETHP to manage storage cells residing in high private storage. This macro instruction provides the same functions as do GETMAIN and FREEMAIN; however, \$GETHP is recommended to increase performance.



## Format description



### TYPE=

Specifies whether the storage cell is to be obtained (GET) or freed (FREE). The default is TYPE=GET.

### CELL=

If TYPE=FREE, then CELL= specifies either the label of a field or a register that contains the address of the storage cell to be freed. If TYPE=GET, then CELL= specifies the register where the address of the storage cell is returned. If CELL= is not specified, register 1 is used.

### FIX=

On a TYPE=GET call, specifies whether page fixed storage (FIX=YES) is requested. The service manages the page fixing and the page freeing of the storage. The default is FIX=NO.

### VERSIZE=

Specifies the length of the requested storage cell. An assembler error occurs if the requested cell length exceeds the length of the cell that can be obtained. The maximum cell length is TRELEN (TCB recovery element length) – 4. TRELEN is defined in the \$TRE macro.

The default is 0.

## Environment

- All environments.
- \$WAIT can occur.
- Callers in AR ASC mode are supported.

## Programming requirements

Be certain to include \$TRE on the \$MODULE macro call.

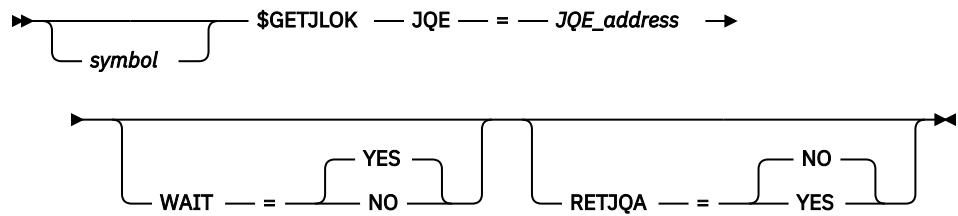
## \$GETJLOK – Acquire the JES2 job lock

Use \$GETJLOK to acquire the JES2 job lock for the specified JQE address.

Obtaining the JES2 job lock (JOB) prevents job queue elements (JQEs) from being changed by any code except the issuer of the job lock.

**Note:** Document in your code the reason for executing the \$GETJLOK macro instruction.

Format description



**JQE=JQE\_address**

Required. Specifies the address of a fullword containing the address of the specified JQE to be acquired in its three right-most bytes. JQE= must be specified for TYPE=JOB.

**WAIT=YES|NO**

Optional. Specifies whether to \$WAIT for the JOB lock to be obtained. This keyword only applies to a TYPE=JOB request; otherwise, it is ignored. The default value is YES.

**RETJQA=YES|NO**

Optional. Specifies whether or not to return the address of the artificial JQE. The default value is NO.

The following register contents apply when \$GETJLOK is invoked:

**Return Code**  
**Meaning**

**0-10**

Not applicable.

**11**

HCT address.

**12**

Not applicable.

**13**

PCE address.

**14-15**

Not applicable.

The following register contents apply on exit from \$GETJLOK:

**Return Code**  
**Meaning**

**0**

Unchanged.

**1**

Address of JQA if RETJQA=YES.

**2-13**

Unchanged.

**14**

Used for linkage.

**15**

Return code:

**0**

LOCK obtained.

**4**

WAIT required.

**8**

JQE represents a different job than originally requested.

## Environment

Only the main task environment can use this macro.

## \$GETLOK – Acquire the MVS CMS, LOCAL, or JES2 job lock

Use \$GETLOK to acquire the MVS CMS, LOCAL, or JES2 job lock depending on the type of lock requested and the environment from which it is requested.

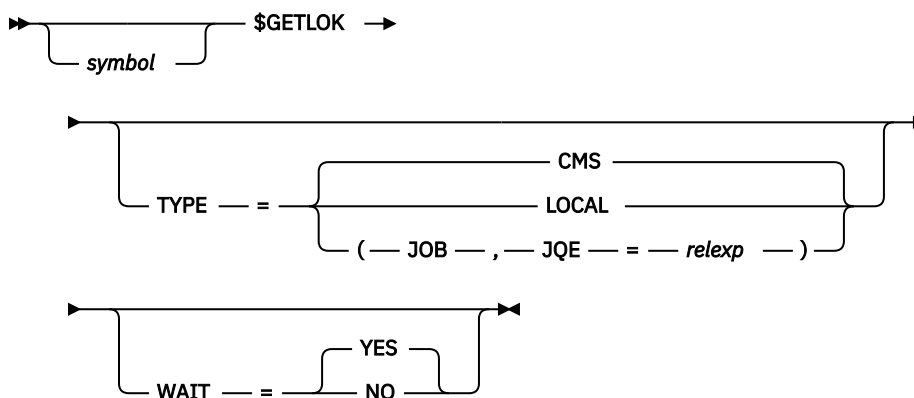
Use \$GETLOK to obtain the cross-memory services (CMS) lock to serialize the JES2 main task with routines that are executing for tasks in other address spaces. The CMS lock is required when modifying certain operating system control blocks and in some cases when interfacing with JES2 code that is running in support of the subsystem interface and access method interface in other address spaces. After obtaining the CMS lock, the user should not execute any code that allows the execution of any SVC instructions until after first freeing the lock through \$FRELOK macro instruction.

Obtaining the local lock (LOCAL) serializes the use of resources such as queues and control blocks among several tasks running within the same address space. The functional subsystem interface (FSI) service routines running in the functional subsystem address space require the local lock for serialization of queues, buffer pools, and control blocks so many separate functional subsystem application (FSA) tasks can use these resources.

Obtaining the JES2 job lock (JOB) prevents job queue elements (JQEs) from being changed by any code except the issuer of the job lock.

**Note:** The reason for executing the \$GETLOK macro instruction should be fully documented in your code.

## Format description



### TYPE=

Specifies the lock to be obtained. Modules assembled for the JES2 environment can specify CMS or JOB only. Modules assembled for the FSS environment can specify CMS or LOCAL only.

#### CMS (default)

The cross-memory lock is to be obtained. All other operands are ignored.

#### LOCAL

The MVS local lock is to be obtained. All other operands are ignored.

#### JOB

The JES2 job lock is to be obtained. In this case a job queue element address (JQE=) must be specified.

### JQE=

Specifies the address of a fullword containing the address of the specified JQE in its three right-most bytes. JQE= must be specified for TYPE=JOB.

## \$GETMLOK

### WAIT=

Specifies whether to \$WAIT for the JOB lock to be obtained. This keyword only applies to a TYPE=JOB request; otherwise, it is ignored. WAIT=YES is the default.

On return from the \$GETMLOK routine, register 15 will contain a return code as follows:

### Return Code

#### Meaning

**0**

Lock obtained

**4**

Wait required

**8**

JQE was freed while waiting for the lock.

## Environment

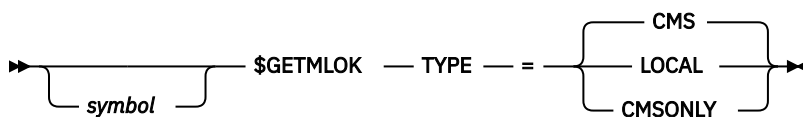
- Main task and functional subsystem (HASPFSSM).
- \$WAIT can occur (if you specify WAIT=YES on the macro).

## \$GETMLOK – Acquire the MVS CMS or LOCAL job lock

Use the \$GETMLOK macro to acquire the CMS or LOCAL lock, or both.

**Note:** Document in your code the reason for executing the \$GETMLOK macro instruction.

## Format description



### TYPE

Specifies the type of lock to manage.

#### LOCAL

Gets the MVS LOCAL lock only.

#### CMS

Gets the CMS lock and the MVS LOCAL lock.

#### CMSONLY

Gets the CMS lock only. This option assumes that the caller already holds the LOCAL lock.

The following register contents apply when \$GETMLOK is invoked:

### Return Code

#### Meaning

**0-10**

Not applicable.

**11**

HCT/HCCT/HFCT address, as applicable.

**12**

Not applicable.

**13**

PCE/Save area address, as applicable.

**14-15**

Not applicable.

The following register contents apply on exit from \$GETMLOK:

**Return Code****Meaning****0**

Modified.

**1-13**

Unchanged.

**14**

Used for linkage.

**15**

Return code:

**0**

LOCAL lock not already held.

**4**

LOCAL lock already held. Always set if TYPE= CMSONLY.

## Environment

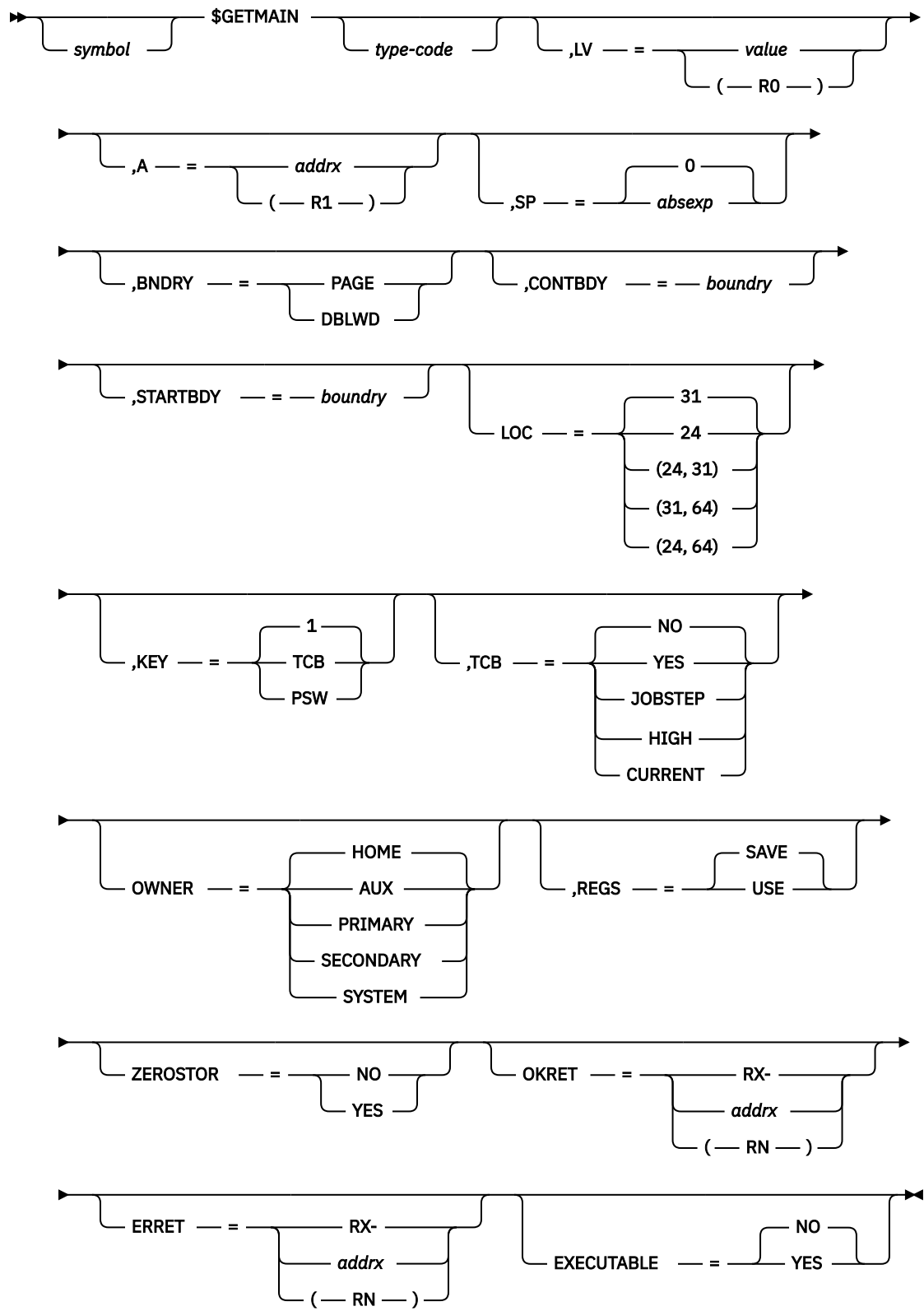
The Main, User, Subtask and FSSM environments can use this macro.

## **\$GETMAIN – Branch-entry GETMAIN services**

---

Use \$GETMAIN in the JES2 main task, user, or subtask environments to obtain an area of storage from MVS GETMAIN/FREEMAIN services or to free an area of storage obtained by this method.

Format Description



**type-code**  
Identifies the type of GETMAIN/FREEMAIN request. The types of requests are defined as follows:

Type	Meaning
------	---------

**R**

An unconditional GETMAIN or FREEMAIN request.

**C**

A conditional GETMAIN request that returns a return code in register 15. (RC=0 indicates a valid GETMAIN, RC≠0 indicates otherwise.)

**RC**

A conditional GETMAIN request that returns a return code in register 15. (RC=0 indicates a valid GETMAIN, RC≠0 indicates otherwise.)

**U**

An unconditional FREEMAIN request. If type-code is not specified on this macro instruction, this is the default.

**RU**

An unconditional FREEMAIN request. In the user's environment, this is the default.

**BC**

A conditional GETMAIN request for a buffer that returns a return code in register 15. (RC=0 indicates a valid GETMAIN, RC≠0 indicates otherwise.) **BC can be used only in the user's environment.**

**BU**

An unconditional FREEMAIN request for a buffer. **BU can be used only in the user's environment.**

**LV=**

Specifies the length of the area to be obtained or freed. This value is loaded into register 0. When this value is coded by way of register format, the subpool can be specified in the high-order byte of the register.

**A=**

Specifies the address of the storage area to be freed. This keyword is only valid if FREEMAIN=YES is also specified.

**SP=**

Identifies the subpool number. Subpool zero is the default if no subpool is specified or the subpool is not specified in the high-order byte of the LV= parameter. This parameter must be specified if you want to free an entire subpool. (In that case, do *not* code the A= parameter.)

**BNDRY=**

Specifies that the storage requested be on either a page or doubleword boundary. This keyword is ignored if the type-code you specify indicates a FREEMAIN request.

**PAGE**

Indicates that the storage is on a 4096-byte (page) boundary.

**DBLWD**

Indicates that the storage is on a doubleword boundary.

**CONTBDY=**

Specifies the boundary that the obtained storage must be contained within. Specify a power of 2 that represents the containing boundary. Supported values are 3-31. For example, CONTBDY=10 means the containing boundary is 2\*\*10, or 1024 bytes. The containing boundary must be at least as large as the maximum requested boundary. The obtained storage does not cross an address that is a multiple of the requested boundary.

If a register is specified, the value must be in bits 52-63 (24-31) of the register.

If you omit this parameter, there is no containing boundary.

**STARTBDY=**

Specifies the boundary the obtained storage must start on. Specify a power of 2 that represents the start boundary. Supported values are 3-31. For example, STARTBDY=10 means the start boundary is 2\*\*10, or 1024 bytes. The obtained storage begins on an address that is a multiple of the requested boundary.

If a register is specified, the value must be in bits 52-63 (24-31) of the register.

If you omit this parameter, the start boundary is 8 bytes (equivalent to specifying STARTBDY=3).

**KEY=**

Specifies the key of the storage that is either to be acquired or freed. If you omit this keyword, JES2 uses, KEY=.

**Note:** Applies only to the user environment when requesting USER/CALLER key storage.

TCB indicates to use the TCBPFK key of the current TCB.

PSW indicates to use the current PSW storage key.

**LOC=**

Specifies the location of the virtual storage to be allocated.

**Note:** Applies only to the JES2 main task environment.

**24**

Indicates that the storage is to be located below the 16-megabyte line.

**31**

Indicates that the storage can be either located above or below the 16-megabyte line in 31-bit storage. This is the default for storage requests in subpools 0-127 in the JES2 main task environment.

**(24, 31)**

Indicates virtual storage is to be located below the 16-megabyte line but the real storage that backs it can be above the 16-megabyte line (in 31-bit storage).

**TCB=**

Specifies what TCB to associate with the storage to be obtained.

**Note:** Applies only to the user environment.

**YES**

Indicates that the TCB address is passed in the first word of the storage to be freed. This form is only valid with an internal request from the \$FREMAIN macro.

**NO (default)**

Indicates a high TCB in the address space if USER key storage is requested. Otherwise, the current or job step TCB is used.

**JOBSTEP**

Indicates the job step TCB.

**HIGH**

Indicates a high TCB in the address space.

**CURRENT**

Indicates the current TCB.

**OWNER=**

For CSA subpools, this indicates the owner assigned to this storage. This is for CSA tracking purposes only and does not affect when the storage is freed. The following specifications are valid:

**AUX**

Associate storage with the JES2 AUX address space.

**HOME**

Associate storage with the home address space. This is the default.

**PRIMARY**

Associate storage with the primary address space.

**SECONDARY**

Associate storage with the secondary address space.

**SYSTEM**

This storage should not be associated with any address space.





**\$GETRTN**

**(R0)**

Specifies that the register contains the quick cell type-code as defined in the \$QCTGEN macro, the two low-order bytes must contain the type-code and the two high-order bytes must be zeroed.

**Note:** The TYPE= keyword must be specified.

**NUM=**

Specifies the number of quick cells to get from the quick cell pool. The value assigned to NUM= must not exceed the specification of QCTLIMIT; exceeding this value will cause an error condition.

***nnn***

Specifies the number (1-255) of quick cells. The default value is 1.

**(Rn)**

Specifies that register notation is used. The register specified contains the number of quick cells.

**STACK=**

Specifies whether the quick cells should be pushed onto a stack or chained together. The chaining field offset is specified in the QCT.

**YES**

Specifies that the quick cells specified by this macro are pushed onto a stack identified by the QCT.

**NO**

Specifies that the quick cells specified by this macro are chained together and the address of the head of the chain is passed back to the caller.

**Note:** Register 11 must contain the address of the HASP function control table (HFCT) before executing \$GETQC.

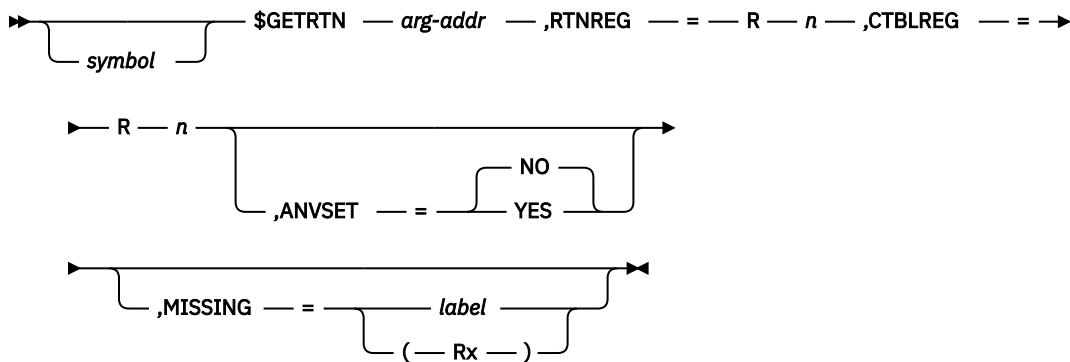
## Environment

- Functional subsystem (HASPFSM).
- MVS WAIT can occur.

## \$GETRTN – Get the address of a routine

\$GETRTN obtains the address of a requested routine. \$GETRTN searches the various assembly environments' routine tables to find the requested routine before returning the address of a local routine. The macro resolves any linkage requirements needed by the routine in a specific environment. Use \$GETRTN to obtain a routine address to pass, through a \$SQD, to a general purpose subtask.

## Format description

**arg-addr**

Use 'arg-addr' to specify one of the following arguments (JES2 checks for defined symbols and processes these arguments in the order listed):

1. A register containing the address of the requested routine.

2. The name (label) of a routine listed in one of the following tables:

**Table**  
**Address Prefix**

**CADDR**  
C@

**PADDR**  
P@

**UCADDR**  
UC@

**UPADDR**  
UP@

JES2 will search through the tables in the above order.

**Note:**

- a. The UPADDR is chained out of \$UPADDR in the \$HCT.
  - b. The UCADDR is chained out of the CCTUCADD in the \$HCCT.
3. • A name (label) of a local routine this macro will call; this causes \$GETRTN to generate an ADCON.
- A label of a field containing the address of a routine this macro will call.
4. The name of a routine contained in another module; this causes \$GETRTN to generate a VCON.

**RTNREG=Rn**

\$GETRTN places the address of the requested routine into this register.

**CTBLREG=Rn**

\$GETRTN uses this register as a base to index into the \$CADDR, \$PADDR, \$UCADDR, and \$UPADDR during the routine address search. Because \$GETRTN overwrites the contents of this register during the address search, do not specify a register that your module needs after invoking the routine. You can use the same register for CTBLREG= as specified in RTNREG=.

**ANVSET=YES|NO**

ANVSET specifies whether the assembly environment should be changed to match the environment of the routine address table in which the routine name was found.

If you change the assembly environment, make sure the calling routine can execute in the new environment.

**Default:** NO.

**MISSING**

Specifies a label to be branched to or a register to be branched on if the address of the routine is zero, or if the address of any control block that is required to get the routine address is zero.

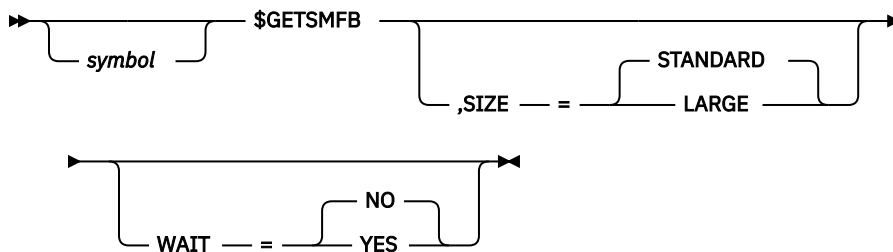
## Environment

- All environments.
- MVS and \$WAIT will not occur.

## \$GETSMFB – Acquire a JES2 SMF buffer from the JES2 SMF buffer pool

Use \$GETSMFB to obtain a buffer from the JES2 SMF buffer pool, clear the buffer contents to binary zeros, and return the address of this buffer in register 1. The macro returns condition code 0 and a 0 in register 1 if no buffers were available and WAIT=NO was specified in the macro.

## Format description



### **SIZE=**

Specifies the size of the SMF buffer to be obtained.

#### **STANDARD**

Indicates that the 920-byte standard size buffer be obtained. This is the default for the SIZE= parameter.

#### **LARGE**

Indicates that a 32K-byte SMF buffer be obtained.

### **WAIT=**

Specifies the action to be taken if no JES2 SMF buffers are available as follows:

#### **YES**

Control is not returned to the caller until a JES2 SMF buffer has become available.

#### **NO**

An immediate return is made. If no buffers are available, register 1 contains a 0 on return to the calling routine. The condition code is nonzero if a buffer is available or 0 if no buffers are available.

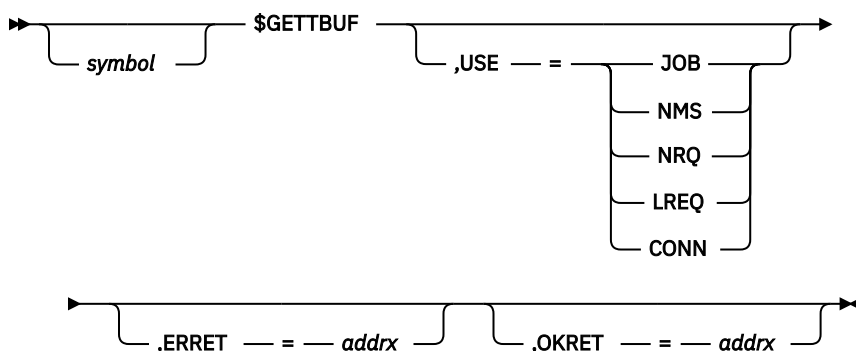
## Environment

- JES2 Main task.
- \$WAIT can occur (if you specify WAIT=YES on the macro).

## \$GETTBUF – Get TCP buffer

Use \$GETTBUF to get a TCP buffer for communication between the JES2 and NETSRV address spaces.

## Format description



### **USE**

Specifies what the buffer will be used for. Valid values are JOB, NMS, NRQ, LREQ, and CONN.

See TBFTYPE for details about what each of these types represents. This parameter is required.

### **ERRET=**

Indicates where to go if no buffer could be obtained.

**OKRET=**

Indicates where to go if a buffer was obtained.

**Environment**

JES2 address space or NETSRV address space.

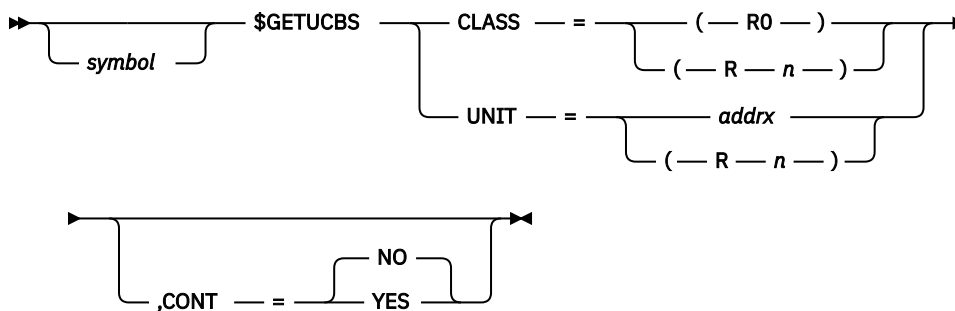
**Return codes**

Return Code	Meaning
-------------	---------

<b>0</b>	Buffer obtained
----------	-----------------

**\$GETUCBS – Obtain a UCB address**

Use \$GETUCBS to obtain a single UCB address or a series of device class UCB addresses, one at a time. JES2 returns the UCB address in the UPLUCB field of the UCB services parameter list (UPL). Register 1 points to the UPL.

**Format description****CLASS=**

Specifies the device class of the UCB or UCBs that are requested.

**UNIT=**

specifies the address of a 4 byte field that contains the EBCDIC unit address of the device corresponding to the requested UCB. The 4 byte field must be on a full word boundary. The variable *addrx* can be either the address of the 4 byte field or a register (registers 2-12, specified in parenthesis) that contains the address of the 4 byte field.

**CONT=YES | NO**

Indicates whether the UCB to be located is (YES) or is not (NO) the UCB for the next device in a series of devices. CONT=NO is the default. **Register 1 must be preserved in the loop when CONT=YES.**

**Return codes**

The following return codes (in decimal) are returned in register 15.

Return Code	Meaning
-------------	---------

<b>0</b>	Processing was successful. Register 1 points to the UPL and the UPLUCB field contains the UCB address.
<b>4</b>	No UCB was found. <ul style="list-style-type: none"> <li>• If you specified UNIT=, you provided a device number for which there is no UCB.</li> </ul>

- If you specified CLASS=, there are no more UCBs for the specified device class.

## Programming considerations

- You must code either CLASS= or UNIT= but not both.
- **To obtain one UCB when you know the EBCDIC unit address:**
  - Issue \$GETUCBS with UNIT= specified and CONT= omitted or coded as CONT=NO.
  - If \$GETUCBS returns a return code of 0, after you finish with the UCB issue \$FREUCBS with UNPIN=YES.
  - If \$GETUCBS returns a return code of non-zero, issue \$FREUCBS with UNPIN=YES.
- **To obtain a series of UCBs, one at a time:**
  1. Issue \$GETUCBS with CONT=NO and the CLASS= parameter specified.
  2. If \$GETUCBS returns a non-zero return code, go to step “8” on page 208. Otherwise, go to the next step.
  3. If you finish processing the UCB and that is the last UCB you want, issue \$FREUCBS with UNPIN=YES and then go to step “9” on page 208. Otherwise, continue with the next step.
  4. Adjust the UCB address in UPLDEVN as follows:
    - Obtain the device number from the UCBCHAN field of the UCB.
    - Add 1 to the device number and store the result into UPLDEVN in the UPL.
  5. Issue the MVS macro UCBPIN with the UNPIN parameter.
  6. Issue \$GETUCBS with CONT=YES and the CLASS= parameter specified to obtain the next UCB in the series. JES overlays the UPL with information about the newly obtained UCB.
  7. If \$GETUCBS returns a return code of 0, return to step “3” on page 208. Otherwise continue.
  8. Issue \$FREUCBS with UNPIN=NO.
  9. Continue with your program.

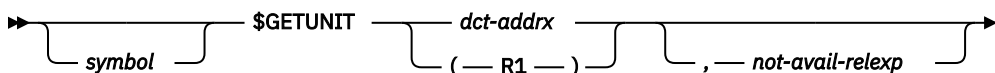
## Environment

- Main task or during JES2 initialization and termination.
- \$WAIT cannot occur.

## \$GETUNIT – Acquire a unit device control table (DCT)

Use \$GETUNIT to assign a device control table (DCT) to a specific device.

## Format description



### dct

Specifies either a pointer to a DCT or the address of a DCT to be obtained. If *dct* is written as an address, then it represents the address of a full word containing the address of the DCT to be obtained. If *dct* is written using register notation (either regular or special register notation), then it represents the address of the DCT to be obtained. If register notation is used, the address must be loaded into the designated register before the execution of the macro instruction. **DCT address must be specified.**

### not-avail

Specifies a location to which control is returned if the specified DCT is not available. If this operand is omitted, the condition code is set to reflect the availability of a DCT as follows:

**CC=0**

The DCT is not available.

**CC≠0**

R1 contains the address of the available DCT.

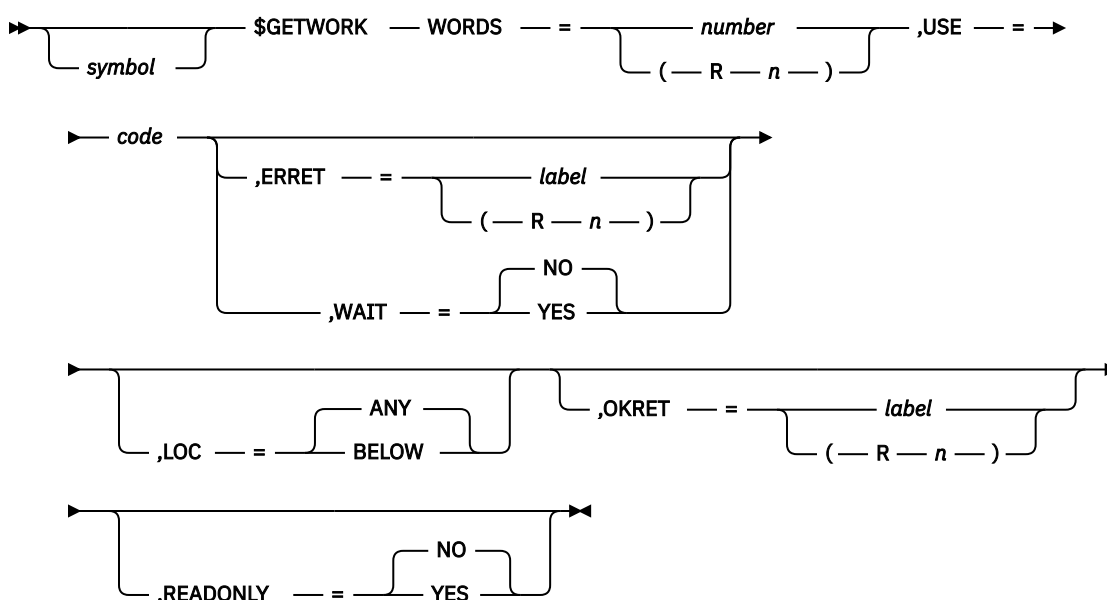
## Environment

- Main task.
- \$WAIT cannot occur.

## \$GETWORK – Obtain a work area

Use \$GETWORK to obtain a work area in subpool 1 in the JES2 address space. If the size of the requested area is appropriate, storage is allocated from JES2-maintained storage pools.

## Format description

**WORDS=**

Specifies the size of the work area in full words or a register that contains the size of the work area in full words.

**USE=**

Specifies the 4-character identifier to be placed in the first four bytes of the work area.

JES2 obtains and clears the storage and then places the identifier into the first four bytes of the work area. Register 1 contains the address of the first byte of the storage area.

JES2 issues catastrophic error \$GW1 if the size requested on the WORDS= operand is greater than the largest size work area supported.

**ERRET=**

Specifies the label or register (R2-R12) that indicates where to branch if \$GETWORK cannot successfully allocate required storage. If ERRET= is not specified or if the allocate fails for any reason, \$GETWORK issues the catastrophic ABEND, GW1. If ERRET is specified, \$GETWORK issues a catastrophic abend for internal errors only, for other errors, (e.g., a GETMAIN failure) register 15 returns a return code of 4. This keyword is not valid if WAIT=YES is also specified.

**WAIT**

Specifies whether (YES) or not (NO) the \$GETWORK service routine is permitted to \$WAIT for storage. This keyword is not valid if ERRET= is also specified.

## \$IOERROR

### LOC=

Specifies the location of the virtual and central storage to be allocated.

### BELOW

Indicates that the storage is to be allocated below the 16-megabyte line.

### ANY

Indicates that the storage can either be located above or below the 16-megabyte line.

### OKRET=

Indicates the label or a register (R2-R12) that contains the address of a routine that is to receive control if JES2 can obtain the requested CSA storage cell.



**Attention:** OKRET= is mutually exclusive with WAIT=YES.

### READONLY=

Indicates whether the memory obtained is readonly (YES) (that is, not key 1) or read/write (NO).

## Environment

- Main task.
- \$WAIT can occur if WAIT=YES is specified.

## \$IOERROR – Log input/output error

---

Use \$IOERROR to log an input/output error on the operator's console.

## Format description



### buffer

Specifies either a pointer to a JES2 buffer or the address of the buffer that has been associated with a JES2 input/output error.

If buffer is written as an address, it represents the address of a fullword that contains the address of the buffer in error in its 3 rightmost bytes. If buffer is written using register notation (either regular or special register notation), it represents the address of the buffer in error. If register 1 is used, the address must be loaded into the register before the execution of the macro instruction.

## Environment

- Main task.
- \$WAIT can occur.

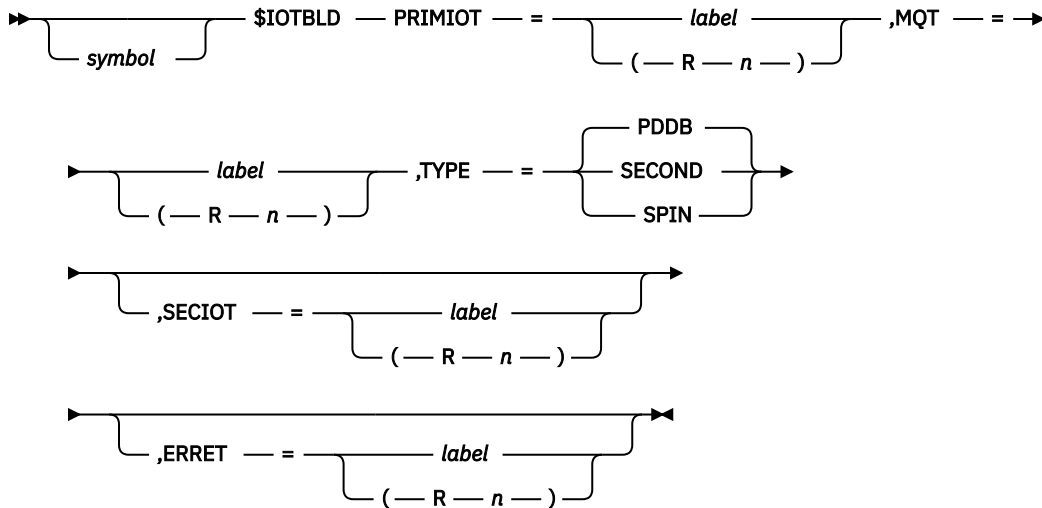
## \$IOTBLD – Build an input/output table (IOT)

---

Use \$IOTBLD to build Pddb-only (peripheral data definition block) IOTs, secondary allocation IOTs, and SPIN IOTs.



## Format Description



### PRIMIOT=

Specifies a label or register that contains the address of the primary allocation IOT.

### MQT=

Specifies the MQT (MTTTT) of the new secondary allocation IOT. This parameter is valid and required only if TYPE=SECOND is specified.

### TYPE=

Specifies the type of IOT to be built and chained into the storage and IOT chains.

#### PDDB

Indicates a PDDB-only IOT

#### SECOND

Indicates a secondary allocation IOT

#### SPIN

Indicates a SPIN IOT

### SECIOT=

Specifies a label or register that contains the address of the buffer into which the secondary IOT is moved. If you specify TYPE=SECOND and do not supply this specification, or the register (or area pointed to) contains zeros, \$IOTBLD still obtains and initializes a buffer as a secondary allocation IOT.

#### Note:

1. This parameter is valid only if TYPE=SECOND is coded.
2. Registers 0 and 2 must not be coded.

### ERRET=

Specifies a label or register that contains the address of the routine that receives control if register 15 contains a nonzero return code.

## Return codes

The following return codes (in decimal) are returned in register 15.

### Return Code

#### Meaning

0

IOT build successful

4

GETMAIN for the IOT buffer failed

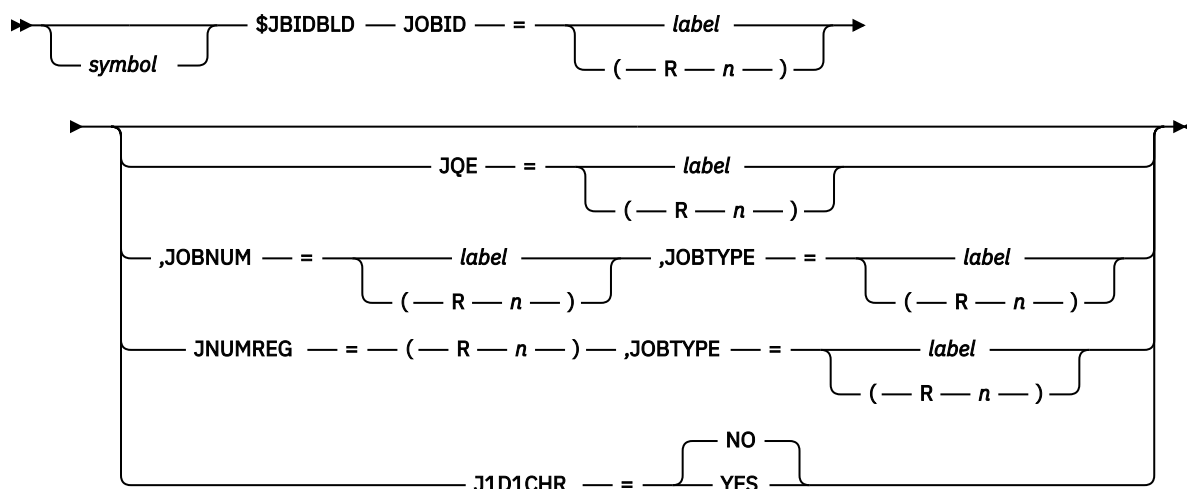
## Environment

- User environment.
- MVS WAIT can occur.

## \$JBIDBLD – Build a JES2 job ID from a binary job number

Use \$JBIDBLD to convert a binary job number to an 8-character JES2 job identifier.

## Format description



### J1D1CHR

Specifies the format of the returned job ID:

#### YES

The returned job ID will have a single character job type prefix and 7 digits of job number.

#### NO

The returned job ID is in a format determined by the maximum job number that is in use in JES2. If the maximum job number that is in use is less than 100000, then a 3-character job type prefix and a 5-digit job number are returned. Otherwise, a 1-character job type prefix and a 7-digit job number is returned. NO is the default

### JOBID=

Specifies a label or a register that contains the address of an 8-byte area where \$JBIDBLD will build the job ID.

### JOBNUM=

Specifies a label or a register that contains the address of a fullword field which contains the binary job number to be converted to a job ID.

### JOBTYPE=

Specifies a label, or a register that contains the address, of the job type flag byte (JOB, STC, TSU) used to construct the job ID. The job type flag byte must be defined. Valid job types, and the bit settings for each are:

#### job type

flag bit setting

#### batch job

xxxxxx00

#### STC

xxxxxx01

**TSU**

xxxxxx10

**JNUMREG=**

A register containing the binary job number that is mutually exclusive with JOBNUM.

**JQE=**

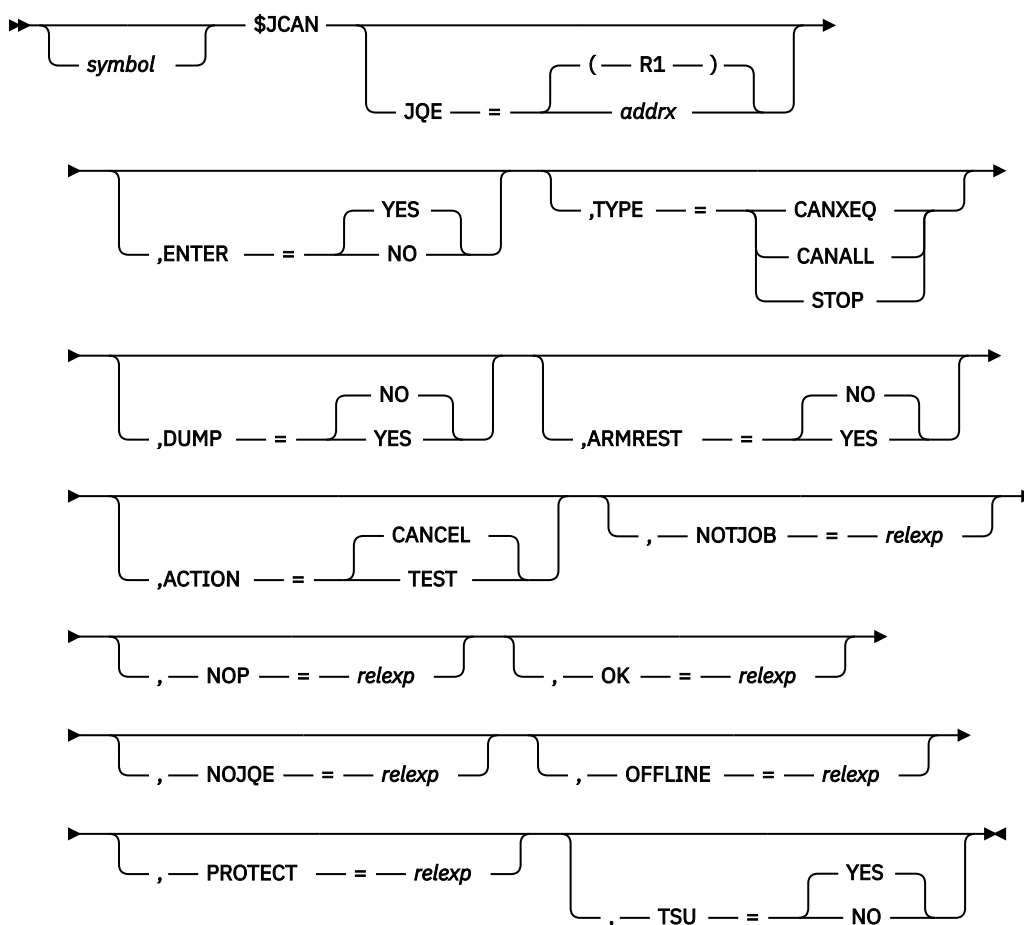
The address of the JQE to be used for data that is mutually exclusive with JNUMREG=, JOBNUM=, and JOBTYP= . The values of the source job number and the jobtype field are derived using the given JQE/JQA address.

**Environment**

- All environments.
- MVS WAIT and \$WAIT cannot occur.

**\$JCAN – Cancel job**

Use \$JCAN to prepare the job represented by the specified job queue element for cancellation of its normal execution and output, or cancellation on completion of its current activity.

**Format description****JQE=**

Specifies the address of the job queue element that represents the job to be canceled. If register notation is used, the address must be loaded into the designated register before the execution of this macro instruction unless ENTER=NO is specified.

**ENTER=**

Specifies whether actual entry to the job cancel service routine is to be affected. If this operand is omitted or YES is specified, the routine is entered. If the specification is NO, the execution of this macro instruction is for setting parameter values in register 0 based on the specifications in the TYPE and DUMP operands. Operands other than ENTER, TYPE, and DUMP should be omitted when ENTER=NO is specified, and the value of register 0 must not be altered until after a later \$JCAN macro with ENTER=YES specified or omitted.

**TYPE=**

Specifies the action to be taken.

**CANXEQ**

Specifies that a normal batch job, which is in the system queues before execution or in execution, is queued for output. A request for a job in \$OUTPUT is considered a no operation, and control is given to the location specified by the NOP operand. If the job is a started task control (STC) or time-sharing user (TSU) job before being executed or in execution, the request is rejected, and control is given to the location specified by the NOTJOB operand.

**CANALL**

The job is canceled from its current activity and queued for purge. If the job is an STC or TSU job before or in execution, the request is rejected, and control is given to the location specified by the NOTJOB operand.

**STOP**

The action is the same as for CANALL except that the job's current activity is not deleted.

If TYPE= specifies a value other than CANXEQ, CANALL, or STOP, the specified value is placed in register 0 (when ENTER=NO), or passed to the service routine in register 0 (when ENTER=YES). If this operand is omitted, register 0 must have been set by a previous execution of a \$JCAN macro instruction specifying ENTER=NO.

**Note:**

1. If the job queue element is currently owned by a processor, queuing to \$OUTPUT or \$PURGE is delayed until the next \$QMOD, \$QPUT, or \$QADD macro instruction is performed.
2. The CANXEQ function may be negated by the execution processor if a re-enqueue function is requested.
3. The CANXEQ function results in cancellation of output if a previous request has been made using the STOP function request.

**DUMP=**

Specifies whether the system is to attempt a storage dump of the specified job whose execution is being canceled. If the specification is DUMP=YES and TYPE=CANXEQ or TYPE=CANALL is specified, and if the job is in execution and is not an STC or TSU job, the system attempts to dump the job in a manner compatible with the MVS CANCEL jobname, DUMP command.

**ARMREST=**

Specifies whether the automatic restart manager is to restart the job after it is canceled, if that job is registered with the automatic restart manager. ARMREST=YES is valid only if TYPE=CANXEQ is also specified.

This parameter is ignored if the job was not in the execution phase of processing or was not registered with the automatic restart manager.

**ACTION=**

ACTION=CANCEL (the default) indicates the job should be canceled, if it is cancellable. ACTION=TEST indicates that the job is to be tested for whether it would be cancellable, and an appropriate return code returned. However, no cancel is attempted.

**NOTJOB=**

Specifies the location to be given control if the job to be canceled is a STC or TSU job in the system before or in execution.

**NOP=**

Specifies the location to be given control if TYPE=CANXEQ is specified and the job has passed the execution phase. If this operand is omitted, control is given to the location specified by the OK operand.

**OK=**

Specifies the location to be given control if the execution of the request is successful. If this operand is omitted, control is given to the location following the macro instruction if the request is successful.

**NOJOE=**

Specifies the location to be given control if the specified number of JOEs were found.

**OFFLINE=**

Specifies the location to be given control if the job's spools is offline.

**Note:** OK=, NOJOE=, OFFLINE= must all be specified in order for any one of them to be recognized.

**PROTECT=**

Specifies the location to be given control if the job's output is protected. If this operand is omitted, no check is made to determine if the job is protected.

**TSU=**

Specifies whether(YES) or not(NO) an active TSU can be canceled. If NO, the rules will be the same as for STCs. If YES, the rules will be the same as for batch jobs. TSU=YES is the default.

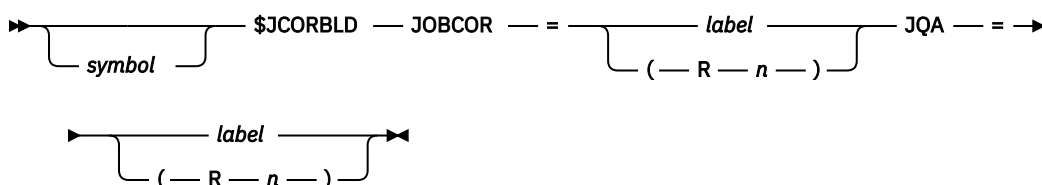
## Environment

- Main task.
- \$WAIT can occur.

## \$JCORBLD – Build a job correlator

Use \$JCORBLD to build a job correlator.

## Format description

**JOB COR=**

Specifies a label or register that contains the address of a 64-character area where \$JCORBLD will build the job correlator. This is a required parameter.

**JQA=**

The address of the JQA to be used to generate the job correlator. This is a required parameter.

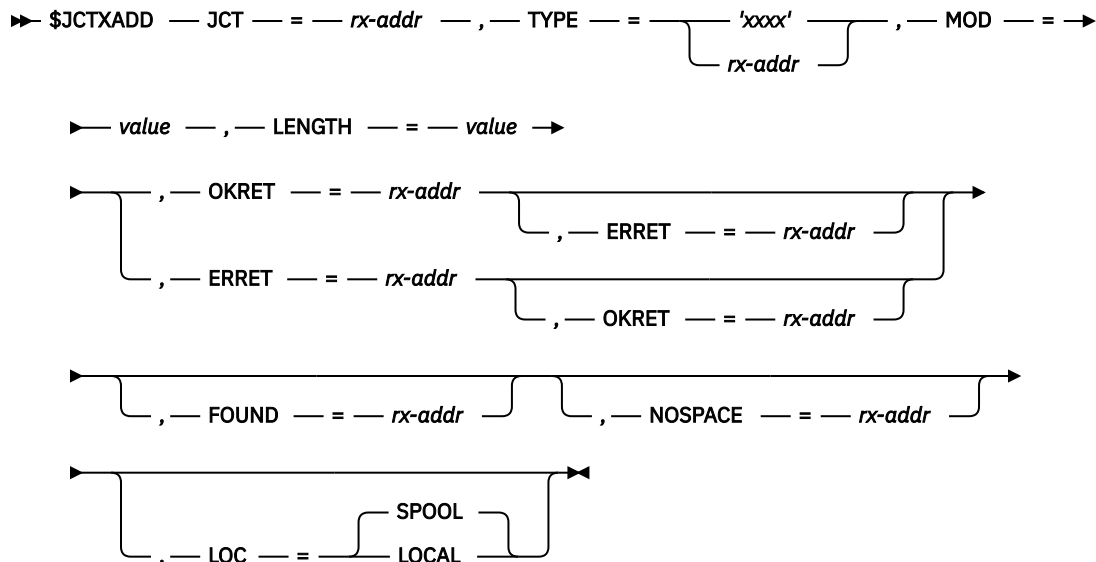
## Environment

- All environments.
- MVS WAIT and \$WAIT cannot occur.

## \$JCTXADD – Add a \$JCT control block extension

Use the \$JCTXADD macro to extend the \$JCT (Job Control Table) control block based on a length and a unique identifier specified by your installation. See [“Using the \\$JCTX macro extension service” on page 10](#) for more information.

## Format description



**JCT=**

Specifies the address of the \$JCT control block to which JES2 should add this extension.

This parameter is required.

**ERRET=**

Specifies the label to receive control if the extension cannot be added for a reason other than those specified through the FOUND= or NOSPACE= parameter (for example, the JCT= parameter does not point to a valid \$JCT control block).

If you do not specify a value for the FOUND= parameter, ERRET= specifies the label to receive control if the extension cannot be added because an extension with a matching type and modifier already exist. If you do not specify a value for the NOSPACE= parameter, ERRET= specifies the label to receive control if the extension cannot be added because the \$JCT is too small.

You must specify the ERRET= parameter, the OKRET= parameter, or both.

**TYPE=**

Specifies either a 1 to 4-character string enclosed in quotation marks ('xxxx'), or the address of a 4-byte field containing such a string, to serve as an identifier for the extension. The strings 'IBM' and 'JES2' are reserved for IBM use.

**Note:** JES2 pads character strings of less than 4 characters with trailing blanks.

**MOD=**

Specifies a value (0-32767) that allows an application to specify a series of extensions to the \$JCT from a single source (through the TYPE= parameter) and to differentiate extensions through this parameter.

MOD= can be specified as:

- A numeric value
- A symbol equated to a value
- A register containing a value
- The address of a 2-byte field containing the value

This parameter is required.

**LENGTH=**

Specifies the length of the extension (0-4095) to be added to the \$JCT control block. This specification must include the length of the \$JCTX prefix area (defined by the value of JCXORG-JCTX field).

The LENGTH= parameter can be specified as:

- A numeric value
- A symbol equated to a value

**Note:** Use an equate to define a field - field value:

```
JCTXULEN EQU    JCTXUEND-JCTX
```

Then specify the value for length in the macro:

```
LENGTH=JCTXULEN
```

If you specify the field - field expression in the macro rather than using an equate to define the expression, JES2 uses the relocatable address from the expression rather than the value.

- A register containing a value
- The address of a 2-byte field containing the length.

The maximum size depends on the:

- \$JCT size (IBM provides a 512-byte spool buffer; any additional space might not be preserved from release to release. To determine the amount of spool used by \$JCT extensions, see [“Using the \\$JCTX macro extension service” on page 10.](#))
- Buffer size (BUFSIZE= parameter on the SPOOLDEF initialization statement)
- Number of extensions already defined in the \$JCT control block.

This parameter is required.

**OKRET=**

Specifies the label to receive control if JES2 adds the extension successfully; **R1** points to the new section on return.

You must specify the OKRET= parameter, the ERRET= parameter, or both.

**FOUND=**

Specifies the label to receive control if an extension with a matching type and modifier already exists in the \$JCT control block; **R1** points to the existing section on return.

If you do not specify this parameter, FOUND= defaults to the value of the ERRET= parameter.

**NOSPACE=**

Specifies the label to receive control if there is insufficient space in the \$JCT control block to add this extension.

If you do not specify this parameter, NOSPACE= defaults to the value of the ERRET= parameter.

**LOC=**

Specifies whether or not the extension should be SPOOLed.

LOC can be specified as:

- SPOOL
- LOCAL

LOC=SPOOL places the extension in the SPOOLed portion of the JCT. LOC=LOCAL places the extension in working storage that is not SPOOLed.

Local extensions are only supported in certain environments. Every extension must have a unique TYPE/MOD irrespective of the extension's location. Default is LOC=SPOOL.

## Return codes

The following return codes (in decimal) are returned in register 15.

<b>Return Code</b>	<b>Meaning</b>
--------------------	----------------

<b>0</b>	JES2 successfully added the extension. <b>R1</b> points to the new extension (corresponding to the OKRET= parameter).
<b>4</b>	JES2 did not add the extension because an extension with the specified type and modifier already exists. <b>R1</b> points to that extension (corresponding to the FOUND= parameter).
<b>8</b>	JES2 did not add the extension because there was insufficient space in the \$JCT control block for the extension. This return code corresponds to the NOSPACE= parameter.
<b>12</b>	JES2 did not add the extension for one of the following reasons: <ul style="list-style-type: none"><li>• An error was detected in the \$JCT control block.</li><li>• An error was detected in the \$JCTX control block.</li><li>• An input parameter was specified incorrectly.</li><li>• Local extensions was not allowed for passed JCT.</li></ul> This return code corresponds to the ERRET= parameter.



## Environment

**Authorization:**

Supervisor state, key 0 or 1.

**Dispatchable unit mode:**

Task

**JES environment:**

Any JES2 environment

**Cross Memory Mode:**

PASN = HASN

**AMODE:**

24- or 31- bit

**ASC mode:**

Primary

**Interrupt status**

Enabled for I/O and external interrupts

**Serialization:**

The \$JCT control block must be serialized; this could mean that the caller has to obtain the job lock or the SJB lock.

**Locks:**

No locks are obtained or freed by the \$JCTXADD macro.

**Control parameters:**

None

## Programming requirements

You must specify \$HCCT, \$JCTX, and \$TRE on the \$MODULE invocation to use this macro.



## Restrictions

None.

## Registers on entry

**R0 - R10:**

N/A

**R11:**

HCT, HCCT, or HFCT, depending on JES2 environment.

**R12:**

N/A

**R13**

Address of PCE or address of an available 72-byte save area

**R14-15:**

N/A

## Registers on exit

**R0:**

Used as a work register by the system.

**R1:**

Pointer to the \$JCT control block extension, or 0.

**R2 - R13:**

Unchanged

**R14**

Used as a work register by the system.

**R15:**

Return code

## Example

```

        $JCTXADD TYPE='USER' ,
                MOD=1,
                JCT=JCT,
                LENGTH=JCXLEN1,
                FOUND=ADDOK,
                ERRET=EXTERR

ADDOK   DS      0H

JCTX     DSECT
        ORG     JCTXORG

JCTXLEN1 EQU    *-JCTX      Installation fields for extension 1
        ORG     JCXORG

JCTXLEN2 EQU    *-JCTX      Installation fields for extension 2
        ORG     JCXORG

```

This macro creates an extension with a type of 'USER' and a modifier of 1. The \$JCT address is in a register with a USING for the \$JCT. The length value must include the following:

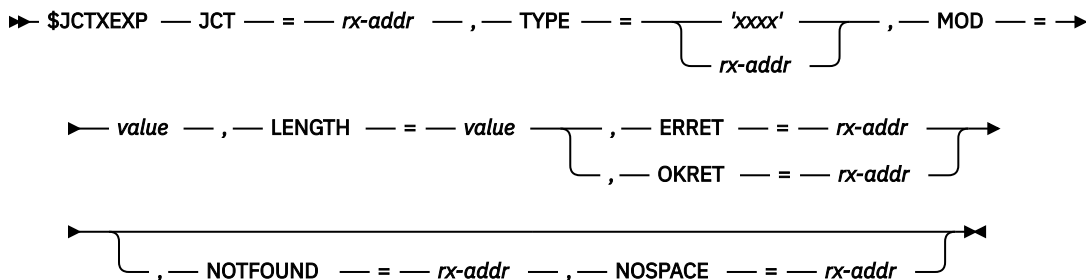
- The JES2-defined header's length (containing an eyecatcher, the ID, and the extension's length)
- The length of any installation fields to be added to the extension. (Begin all installation fields at label JCXORG).

If the extension is successfully added (or already exists in the \$JCT), processing continues at the next sequential instruction. Otherwise, processing continues at label EXTERR.

## \$JCTXEXP – Expand a \$JCT Control block extension

Use the \$JCTXEXP macro to extend the \$JCT (Job Control Table) control block by a length and a unique identifier specified by your installation. You cannot specify a length shorter than the original extension. The extension can be a SPOOLED or a local extension. If you specify a shorter length, JES2 returns the length of the original extension. See [“Using the \\$JCTX macro extension service” on page 10](#) for more information.

### Format description



#### JCT=

Specifies the address of the \$JCT control block to which JES2 should expand this extension.

This parameter is required.

#### ERRET=

Specifies the label to receive control if the extension cannot be expanded for a reason other than those specified through the NOTFOUND= or NOSPACE= parameter (for example, the JCT= parameter does not point to a valid \$JCT control block).

If you do not specify a value for the NOTFOUND= parameter, ERRET= specifies the label to receive control if the extension does not exist. If you do not specify a value for the NOSPACE= parameter, ERRET= specifies the label to receive control if the extension cannot be expanded because the \$JCT is too small.

You must specify the ERRET= parameter, the OKRET= parameter, or both.

#### TYPE=

Specifies either a 1 to 4-character string enclosed in quotation marks ('xxxx'), or the address of a 4-byte field containing such a string, to serve as an identifier for the extension. The strings 'IBM' and 'JES2' are reserved for IBM use.

**Note:** JES2 pads character strings of less than 4 characters with trailing blanks.

#### MOD=

Specifies a value (0-32767) that allows an application to specify a series of extensions to the \$JCT from a single source (through the TYPE= parameter) and to differentiate extensions through this parameter.

MOD= can be specified as:

- A numeric value
- A symbol equated to a value
- A register containing a value
- The address of a 2-byte field containing the value

This parameter is required.

#### LENGTH=

Specifies the total length of the \$JCT control block extension *after* expansion. Specify the total length of the extension. For example, if the original extension was 20 bytes long and this expansion adds an

additional 10 bytes, specify 30 bytes for this parameter. This specification must include the length of the \$JCTX prefix area (defined by the value of the JCXORG-JCTX field).

The LENGTH= parameter can be specified as:

- A numeric value
- A symbol equated to a value

**Note:** Use an equate to define a field - field value:

```
JCTXULEN EQU    JCTXUEND-JCTX
```

Then specify the value for length in the macro:

```
LENGTH=JCTXULEN
```

If you specify the field - field expression in the macro rather than using an equate to define the expression, JES2 uses the relocatable address from the expression rather than the value.

- A register containing a value
- The address of a 2-byte field containing the length.

The maximum size depends on the:

- \$JCT size (IBM provides a 512-byte spool buffer; any additional space might not be preserved from release to release. To determine the amount of spool used by \$JCT extensions, see [“Using the \\$JCTX macro extension service” on page 10.](#))
- Buffer size (BUFSIZE= parameter on the SPOOLDEF initialization statement)
- Number of extensions already defined in the \$JCT control block.

This parameter is required.

#### **OKRET=**

Specifies the label to receive control if JES2 expands the extension successfully; **R1** points to the new section on return.

You must specify the OKRET= parameter, the ERRET= parameter, or both.

#### **NOTFOUND=**

Specifies the label to receive control if the extension could not be expanded because it does not exist; **R1** points to the existing section on return.

If you do not specify this parameter, NOTFOUND= defaults to the value of the ERRET= parameter.

#### **NOSPACE=**

Specifies the label to receive control if there is insufficient space in the \$JCT control block to expand this extension.

If you do not specify this parameter, NOSPACE= defaults to the value of the ERRET= parameter.

## **Return codes**

The following return codes (in decimal) are returned in register 15.

#### **Return Code Meaning**

**0**

JES2 successfully expanded the extension. **R1** points to the new extension (corresponding to the OKRET= parameter).

**4**

JES2 did not expand the extension because an extension with the specified type and modifier already exists. This return code corresponds to the NOTFOUND= parameter.

**8**

JES2 did not expand the extension because there was insufficient space in the \$JCT control block for the extension. This return code corresponds to the NOSPACE= parameter.

**12**

JES2 did not expand the extension for one of the following reasons:

- An error was detected in the \$JCT control block.
- An error was detected in the \$JCTX control block.
- An input parameter was specified incorrectly.

This return code corresponds to the ERRET= parameter.

## Environment

**Authorization:**

Supervisor state, key 0 or 1.

**Dispatchable unit mode:**

Task

**JES environment:**

Any JES2 environment

**Cross Memory Mode:**

PASN = HASN

**AMODE:**

24- or 31- bit

**ASC mode:**

Primary

**Interrupt status**

Enabled for I/O and external interrupts

**Serialization:**

The \$JCT control block must be serialized; this could mean that the caller has to obtain the job lock or the SJB lock.

**Locks:**

No locks are obtained or freed by the \$JCTXADD macro.

**Control parameters:**

None

## Programming requirements

You must specify \$HCCT, \$JCTX, and \$TRE on the \$MODULE invocation to use this macro.

## Restrictions

None.

## Registers on entry

**R0 - R10:**

N/A

**R11:**

HCT, HCCT, or HFCT, depending on JES2 environment.

**R12:**

N/A

**R13**

Address of PCE or address of an available 72-byte save area

**R14-15:**

N/A

**Registers on exit****R0:**

Used as a work register by the system.

**R1:**

Pointer to the \$JCT control block extension, or 0.

**R2 - R13:**

Unchanged

**R14**

Used as a work register by the system.

**R15:**

Return code

**Example**

```
$JCTXEXP TYPE='USER' ,
        MOD=1,
        JCT=JCT,
        LENGTH=JCXLEN1,
        ERRET=EXTERR
```

This macro expands an extension with a type of 'USER' and a modifier of 1. The \$JCT address is in a register with a USING for the \$JCT. The length value should include the following:

- The JES2-defined header's length (containing an eyecatcher, the ID, and the extension's length)
- The length of any installation fields to be added to the extension. (Begin all installation fields at label JCXORG).

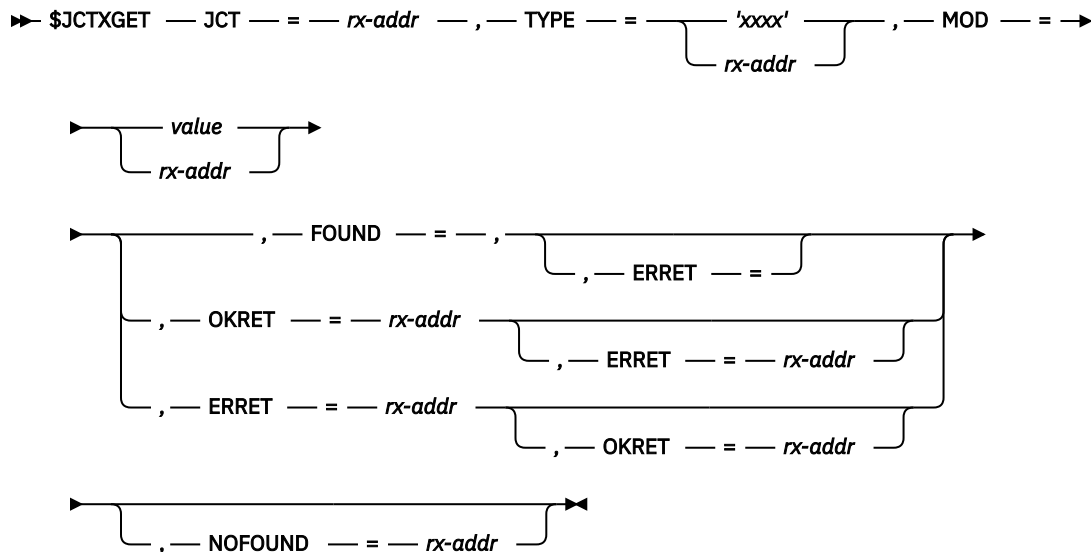
For sample definitions, see [“Example” on page 219](#).

If the extension is successfully expanded, processing continues at the next sequential instruction. Otherwise, processing continues at label EXTERR.

**\$JCTXGET – Get a \$JCT extension**

Given a JCT address, type, and modifier, will locate the JCT extension with that type and modifier in the JCT. SPOOLED JCT extensions are searched first; then the local extensions. The reason code indicates where the extension was found.

## Format description



### JCT=

Specifies the address of the \$JCT control block where this extension resides.

This parameter is required.

### ERRET=

Specifies the label to receive control if the extension cannot be found.

You must specify the ERRET= parameter, the OKRET= parameter, or both.

### TYPE=

Specifies either a 1 to 4-character string enclosed in quotation marks ('xxxx'), or the address of a 4-byte field containing such a string, to serve as an identifier for the extension. The strings 'IBM' and 'JES2' are reserved for IBM use.

**Note:** JES2 pads character strings of less than 4 characters with trailing blanks.

### MOD=

Specifies a value (0-32767) that allows an application to specify a series of extensions to the \$JCT from a single source (through the TYPE= parameter) and to differentiate extensions through this parameter.

MOD= can be specified as:

- A numeric value
- A symbol equated to a value
- A register containing a value
- The address of a 2-byte field containing the value

This parameter is required.

### FOUND=

Specifies the label to receive control if JES2 locates an extension with a matching type and modifier in the \$JCT control block; **R1** points to the section on return.

The FOUND= parameter is mutually exclusive with the OKRET= parameter.

### OKRET=

Specifies the label to receive control if JES2 locates an extension with a matching type and modifier in the \$JCT control block; **R1** points to the section on return.

The OKRET= parameter is mutually exclusive with the FOUND= parameter.

You must specify the OKRET= parameter, the ERRET= parameter, or both.

**NOTFOUND=**

Specifies the label to receive control if an extension with a matching type and modifier does not exist in the \$JCT control block.

If you do not specify this parameter, it defaults to the ERRET= value.

## Return codes

The following return codes (in decimal) are returned in register 15.

**Return Code**

**Meaning**

**0**

Extension found.

**4**

Extension not found.

**8**

Extension not found - error detected.

Reason code (register 0) when the return code in register 15 is zero.

**Reason Code**

**Meaning**

**0**

Extension found in SPOOLed JCT.

**4**

Extension found in local JCT extension area.

## Environment

**Authorization:**

Supervisor state, key 0 or 1.

**Dispatchable unit mode:**

Task

**JES environment:**

Any JES2 environment

**Cross Memory Mode:**

PASN = HASN

**AMODE:**

24- or 31- bit

**ASC mode:**

Primary

**Interrupt status**

Enabled for I/O and external interrupts

**Serialization:**

None

**Locks:**

No locks are obtained or freed by the \$JCTXADD macro.

**Control parameters:**

None

## Programming requirements

You must specify \$HCCT, \$JCTX, and \$TRE on the \$MODULE invocation to use this macro.

## Restrictions

Any data beyond the defined extension should not be referenced or modified.

## Registers on entry

**R0 - R10:**

N/A

**R11:**

HCT, HCCT, or HFCT, depending on JES2 environment.

**R12:**

N/A

**R13**

Address of PCE or address of an available 72-byte save area

**R14-15:**

N/A

## Registers on exit

**R0:**

Used as a work register by the system.

**R1:**

Pointer to the \$JCT control block extension, or 0.

**R2 - R13:**

Unchanged

**R14**

Used as a work register by the system.

**R15:**

Return code

## Example

```
$JCTXGET TYPE='USER' ,  
        MOD=1,  
        JCT=JCT ,  
        ERRET=EXTERR
```

This macro locates an extension with a type of 'USER' and a modifier of 1. The \$JCT address is in a register with a USING for the \$JCT.

Any data beyond the extension should not be referenced or modified. To find the total length of the extension, see the JCXLEN field in each \$JCT extension. If you need to increase the length of an extension to include referenced data, use the \$JCTXEXP macro.

If the extension is successfully located, processing continues at the next sequential instruction. Otherwise, processing continues at label EXTERR.

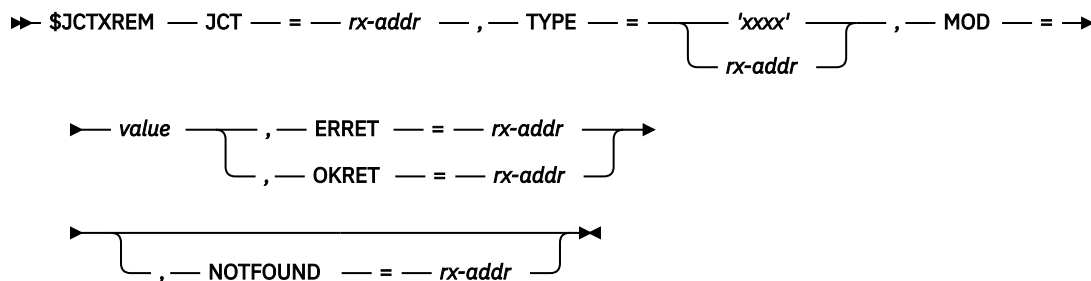
## **\$JCTXREM – Remove a \$JCT control block extension**

---

Deletes a \$JCT extension. The extension can be a SPOOLed or a local extension.



## Format Description



### JCT=

Specifies the address of the \$JCT control block from which JES2 should delete this extension.

This parameter is required.

### ERRET=

Specifies the label to receive control if the extension cannot be deleted for a reason other than those specified through the NOTFOUND= parameter (for example, the JCT= parameter does not point to a valid \$JCT control block).

If you do not specify a value for the NOTFOUND= parameter, ERRET= specifies the label to receive control if the extension cannot be deleted because it does not exist.

You must specify the ERRET= parameter, the OKRET= parameter, or both.

### TYPE=

Specifies either a 1 to 4-character string enclosed in quotation marks ('xxxx'), or the address of a 4-byte field containing such a string, to serve as an identifier for the extension. The strings 'IBM' and 'JES2' are reserved for IBM use.

**Note:** JES2 pads character strings of less than 4 characters with trailing blanks.

### MOD=

Specifies a value (0-32767) that allows an application to specify a series of extensions to the \$JCT from a single source (through the TYPE= parameter) and to differentiate extensions through this parameter.

MOD= can be specified as:

- A numeric value
- A symbol equated to a value
- A register containing a value
- The address of a 2-byte field containing the value

This parameter is required.

### OKRET=

Specifies the label to receive control if JES2 deletes the extension successfully; **R1** points to the new section on return.

You must specify the OKRET= parameter, the ERRET= parameter, or both.

### NOTFOUND=

Specifies the label to receive control if an extension with a matching type and modifier is not found in the \$JCT control block.

If you do not specify this parameter, NOTFOUND= defaults to the value of the ERRET= parameter.

## Return codes

The following return codes (in decimal) are returned in register 15.

## Return Code Meaning

**0**

JES2 successfully deleted the extension. The result corresponds to the macro's OKRET= parameter.

**4**

JES2 did not delete the extension because a section with the specified type and modifier does not exist. This return code corresponds to the NOTFOUND= parameter.

**8**

JES2 did not delete the extension for one of the following reasons:

- An error was detected in the \$JCT control block.
- An error was detected in the \$JCTX control block.
- An input parameter was specified incorrectly.

This return code corresponds to the ERRET= parameter.

## Environment

### Authorization:

Supervisor state, key 0 or 1.

### Dispatchable unit mode:

Task

### JES environment:

Any JES2 environment

### Cross Memory Mode:

PASN = HASN

### AMODE:

24- or 31- bit

### ASC mode:

Primary

### Interrupt status

Enabled for I/O and external interrupts

### Serialization:

The \$JCT control block must be serialized; this could mean that the caller has to obtain the job lock or the SJB lock.

### Locks:

No locks are obtained or freed by the \$JCTXREM macro.

### Control parameters:

None

## Programming requirements

You must specify \$HCCT, \$JCTX, and \$TRE on the \$MODULE invocation to use this macro.

## Restrictions

None.

## Registers on entry

### R0 - R10:

N/A

### R11:

HCT, HCCT, or HFCT, depending on JES2 environment.

**R12:**

N/A

**R13**

Address of PCE or address of an available 72-byte save area

**R14-15:**

N/A

**Registers on exit****R0 - R1:**

Used as work registers by the system.

**R2 - R13:**

Unchanged

**R14**

Used as a work register by the system.

**R15:**

Return code

**Example**

```
$JCTXREM TYPE='USER',
        MOD=1,
        JCT=JCT,
        ERRET=EXTERR
```

This macro deletes an extension with a type of 'USER' and a modifier of 1. The \$JCT address is in a register with a USING for the \$JCT.

If the extension is successfully deleted, processing continues at the next sequential instruction. Otherwise, processing continues at label EXTERR.

**\$JQEJNUM – Obtain JQE job number**

\$JQEJNUM generates in-line code to obtain the job number from a specified JQE. The macro determines the JES2 checkpoint mode and which JQE fields to use.

**Format description**

$\text{symbol} \quad \$JQEJNUM \text{ --- } , \text{ --- } JQE \text{ --- } = \text{ --- } ( \text{ --- } R \text{ --- } n \text{ --- } ) \text{ --- } , \text{ --- } REG \text{ --- } = \text{ --- } \rightarrow$ 
  
 $\text{--- } R \text{ --- } x \text{ ---}$

**JQE=**

The JQE from which to obtain the job number.

**REG=**

The register in which JES2 places the returned job number.

**Return codes**

None.

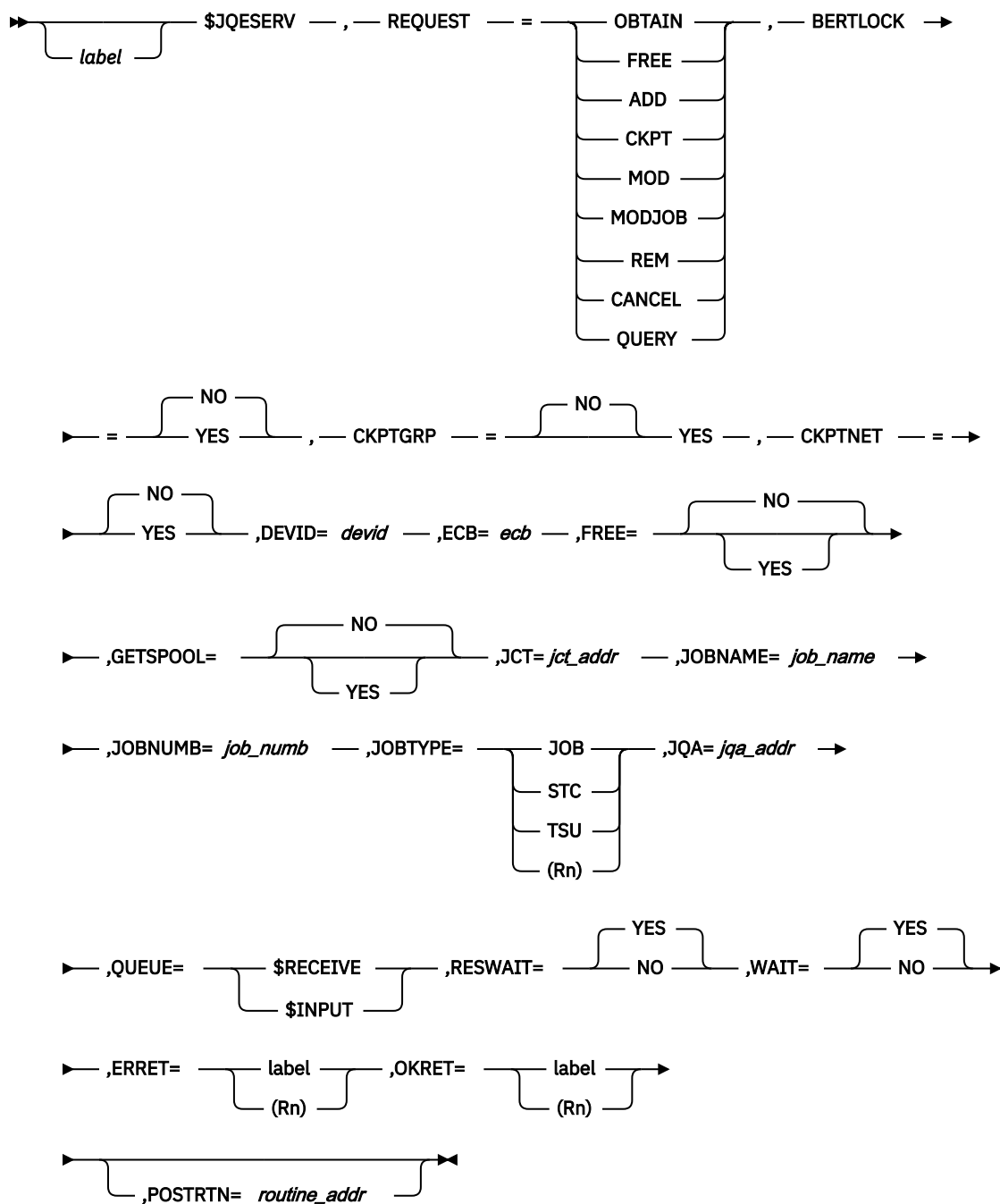
**Environment**

- JES2 main task

## \$JQESERV – User environment JQE services

Use \$JQESERV to request JQE services for code that is not running in the main task.

### Format description



#### REQUEST=

Specifies the function being requested. This is a required operand. Valid requests include the following:

##### OBTAIN

Get storage for a JQA that can be later passed in to this service as JQA=.

##### FREE

Free storage that is previously obtained for a JQA.

**ADD**

Obtain a JQE, job number, and job key and add it to the job queue. Optionally, a JCT and an IOT can be obtained. Input to ADD can be either a JQA with the appropriate fields set or the individual fields that are passed as parameters.

**CKPT**

Cause a \$CKPT of the JQE passed.

**MOD**

Requeue a JQE to the queue passed. This request also releases the BERT lock (if obtained by ADD) and frees the JQE.

**REM**

Unconditionally make the job go away. This does not attempt to free any SPOOL space the job might have. It is intended for recovery paths where the state of the JQE might not be known.

**CANCEL**

Cancel any requests that are associated with the current task. No other operands are allowed with CANCEL.

**QUERY**

Determine whether a JQRB for the specified JQE exists. Only the JQA operand is allowed with QUERY to specify the JQE.

**BERTLOCK=**

For REQUEST=ADD, BERTLOCK= specifies whether the job should be returned with the BERT lock held (BERTLOCK=YES) or not (BERTLOCK=NO). This is an optional parameter on REQUEST=ADD. The default is BERTLOCK=NO.

**CKPTGRP=**

For REQUEST=CKPT, denotes that CKPT is job group specific and JQA passed is a logging job. JQXZNAME contains ZOD address on input and job group name on successful exit. The default is CKPTGRP=NO.

**CKPTNET=**

For REQUEST=CKPT, denotes that CKPT is //\*NET specific and JQA passed is the job on the job card that is associated with the //\*NET. JQXZNAME contains JDBLDWRK work area that is associated with the //\*NET. On successful exit contains job group name in support of JEC network. The default is CKPTNET=NO.

**DEVID=**

For REQUEST=ADD, DEVID= points to the 3 byte device id to be placed in JQEDEVID. JES2 always marks JQEs busy. This parameter is required for REQUEST=ADD, if JQA= is not specified.

**ECB=**

When WAIT=NO, ECB= specifies the ECB to post when the request completes. This parameter is incompatible with WAIT=YES. This is an optional parameter for REQUEST types ADD, CKPT, MOD, REM, and MODJOB.

**FREE=**

Specifies whether (YES) or not (NO) the JQA should be freed after processing the request. This is an optional parameter. The default is NO. The parameter is only valid when JQA= is specified. This is an optional parameter for REQUEST types CKPT, MOD, and REM.

**GETSPOOL=**

Specifies whether (YES) or not (NO) a JCT and an IOT should be obtained for the job and returned to the caller. GETSPOOL=YES builds a basic JCT and primary allocation IOT, obtains SPOOL space for them, and starts an initial write of the data areas. This is recommended so that the JQE can anchor any SPOOL space that is allocated to the job. This is an optional parameter on REQUEST=ADD. The default is GETSPOOL=YES. GETSPOOL=NO is not allowed if a JCT was passed.

**JCT=**

Specifies the address of a JCT that is written by the service. The parameter is not valid if GETSPOOL=NO. The job key, job number, JQE offset, and IOT track address is set by the service before writing the JCT. This parameter is optional for REQUEST=ADD. For REQUEST=MOD, this is the JCT that is passed through \$QMOD to exit 51 (optional parameter).

**JOBNAME=**

Specifies pointer to the 8 byte job name to be assigned to the job. This parameter is required for REQUEST=ADD if JQA= is not specified.

**JOBNUMB=**

Specifies optional job number to be assigned to the job if available. This is the value for JQXIJNUM. This is an optional parameter on REQUEST=ADD if JQA= is not specified. The default is to assign the next available job number.

**JOBTYP=**

Specifies type of job to be added to the job queue (JOB/STC/TSU) or a register with 0, 1, 2 (0, JQE3STC, JQE3TSU) indicating JOB, STC, or TSU, respectively. This is an optional parameter on REQUEST=ADD if JQA= is not specified. The default is JOBTYP=JOB.

**JQA=**

Specifies the address of the JQA to be processed. For REQUEST=ADD, either a JQA with JQENAME, JQETYP, JQE3JOB, and JQEDEVID set in the past JQA, or the individual parameters set. This operand is required for all other request types.

**MJPARMS**

Modify Job parameter list. Required when REQUEST type MODJOB is specified. This is a 4 byte address of the CSA storage location of the Modify Job parameter list. This parameter may be specified using register notation (R2 thru R12) or as an *rx-addr* expression.

**POSTRTN**

Specifies the address of the invoker's post exit routine that is called when a JQESERV request completes. This call is made prior to posting any supplied ECB. The invoker's post exit routine can perform request-specific processing. The return code from this routine also determines if any supplied ECB is posted. A return code of 0 causes the ECB to be posted, if supplied. A return code greater than 0 causes the JQESERV post exit processing to bypass posting any supplied ECB. In this manner, an invoker can logically group several JQESERV requests together, and only allow the ECB to be posted when all requests are completed. This is an optional parameter for REQUEST types ADD, CKPT, MOD, REM, and MODJOB.

**QUEUE=**

For REQUEST=ADD and REQUEST=MOD, this specifies the queue to place the JQE on. This is the 1 byte JQETYP value that represents the queue. This parameter is optional for REQUEST=ADD. The default is based on the device type (SYSOUT NJE devices use \$RECEIVE, all other device types use \$INPUT). This parameter is required for REQUEST=MOD.

**RESWAIT=**

Specifies whether the code should wait for resources (YES) or return an error return code (NO) if resources are not available (JQEs, BERTs, JOB numbers, SPOOL space) for a REQUEST=ADD. The default is RESWAIT=YES.

**WAIT=**

Specifies whether (YES) or not (NO) the caller wants to wait for the request to complete. WAIT=YES is incompatible with ECB=. WAIT=NO returns to the caller before the request completes. This is an optional parameter for REQUEST types CKPT, MOD, and REM. WAIT=NO is required for REQUEST type MODJOB. The default is WAIT=YES.

**ERRET= (optional)**

Specifies a label to be branched to or a register to be branched on if a non-zero return code is returned in R15.

**OKRET= (optional)**

Specifies a label to be branched to or a register to be branched on if a zero return code is returned in R15.

Table 6. \$JQESERV Parameter Table									
Parameter	ADD	CKPT	MOD	REM	OBTAIN	FREE	CANCEL	QUERY	MODJOB
BERTLOCK	O	X	X	X	X	X	X	X	X
DEVID	Oa	X	X	X	X	X	X	X	X

Table 6. \$JQESERV Parameter Table (continued)

Parameter	ADD	CKPT	MOD	REM	OBTAIN	FREE	CANCEL	QUERY	MODJOB
ECB	O	O	O	O	X	X	X	X	O
FREE	X	O	O	O	X	X	X	X	X
GETSPOOL	O	X	X	X	X	X	X	X	X
JCT	O	X	O	X	X	X	X	X	X
JOBNAME	Oa	X	X	X	X	X	X	X	X
JOBNUMB	Oa	X	X	X	X	X	X	X	X
JOBTYPE	Oa	X	X	X	X	X	X	X	X
JQA	Ob	R	R	R	X	R	R	R	X
POSTRTN	O	O	O	O	X	X	X	X	O
QUEUE	O	X	R	X	X	X	X	X	X
RESWAIT	O	X	X	X	X	X	X	X	X
WAIT	X	O	O	O	X	X	X	X	R
MJPARMS	X	X	X	X	X	X	X	X	R
CKPTGRP	X	O	X	X	X	X	X	X	X
CKPTNET	X	O	X	X	X	X	X	X	X
Keys: X = Not valid O = Optional R = Required ADD Calling Options a – set 1 of options b – set 2 of options									

## Environment

- JES2 USER environment.

## Registers on entry

### R0 - R10:

N/A

### R11:

HCT based address.

### R12:

N/A

### R13

PCE base address.

### R14 - R15:

N/A

## Registers on exit

### R0:

JCT address

### R1:

JQA address

### R2 - R13:

Unchanged

### R14

IOT address

**R15:**  
Return code

**Return codes**

The following return codes (in decimal) are returned in register 15:

Return Code	Meaning
0	Processing successful.
4	Resources unavailable (ADD) or JQRB not found for JQE (QUERY).
8	JQE or JQA passed not found (MOD, CKPT, REM).
12	Severe error encountered.

**\$LOGMSG – Log a job-related message**

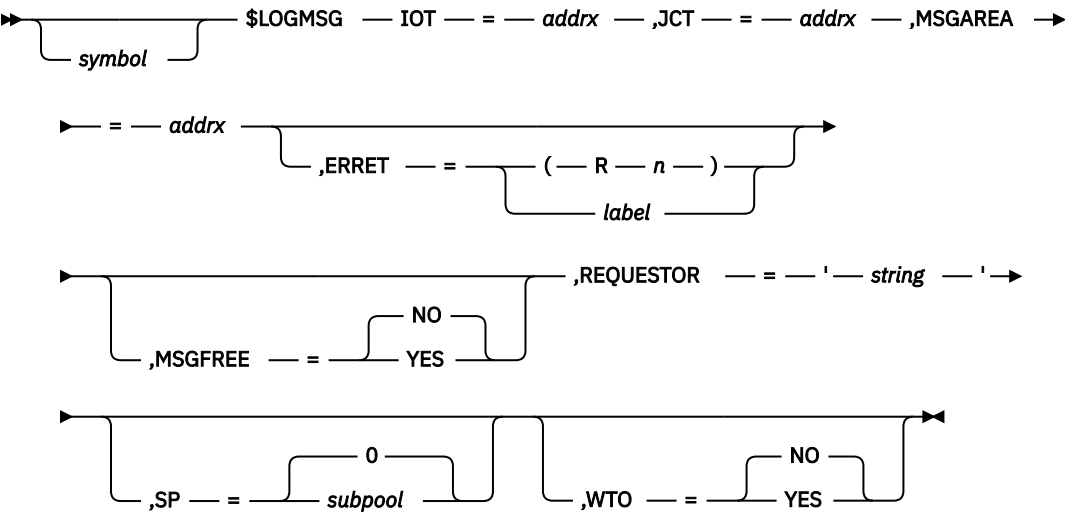
\$LOGMSG places job-related messages into the job's JOBLLOG data set and optionally writes the messages to the operator using WTO. Invoke this macro during any phase of job processing except conversion and execution.

If issuing this macro from JES2 installation exit 4, you must first issue a \$SUBIT call to request that the subtask allow \$LOGMSG to run under that subtask. \$LOGMSG cannot be used in Exits 2 and 3.

When using this macro, you must:

- Ensure that the JCT and primary allocation IOT addresses are non-zero.
- Ensure that the JCTTRAK and IOTTRACK fields are non-zero.
- Construct the message chain.
- Ensure that the job is not in conversion or execution phase.

**Format description**



**IOT=**  
This is the address of the job's primary allocation IOT. IOT= is required.



**JCT=**

JCT= specifies the address of the JCT that represents the job associated with the messages. JCT= is required.

**MSGAREA=**

MSGAREA= specifies a pointer to the starting address of a chain of messages JES2 places into the job-related data set. The messages are in WTO parameter list format which has a two fullword header.

The header consists of:

**Word 1**

Length of the entire WTO parameter list including the message text plus the 8-byte prefix to the text.

This field specifies the length of the area to be freed when MSGFREE=YES is specified.

**Word 2**

Address of the next message or 0.

MSGAREA= is required.

**ERRET=(Rn)|label**

Specifies the label of, or a register that contains, the address of a routine that receives control if the operation was unsuccessful (that is, if register 15 contains a nonzero return code).

**MSGFREE=NO|YES**

MSGFREE= specifies whether JES2 frees the message areas after processing. If you code MSGFREE=YES, you must also code SP=. The default for MSGFREE is NO.

**REQUESTOR='string'**

Specifies the 1 to 255-character identifier of the caller of this macro. JES2 adds this information to your message to provide additional information about the origin of the message. REQUESTOR= is required and must be enclosed in single quotation marks.

**Note:** This information is currently not used by JES2; rather it is used to provide a reason for issuing the message, such as: security authorization failed for job validation.

**SP=0|subpool**

SP= represents the subpool in which the message areas are located. You must code SP= if you coded MSGFREE=YES. The value of SP= must be a literal value or an equate. SP= defaults to 0.

**WTO=NO|YES**

Specifies whether to write the messages to the operator using WTO. The default is NO.

## Return codes

The following return codes (in decimal) are returned in register 15.

**Return Code****Meaning**

**0**

Processing successful - no errors.

**4**

Unable to open the JOBLLOG data set.

## Environment

- JES2 subtask.
- MVS WAITs can occur.

## **\$MID – Assign JES2 message identification**

---

Use \$MID to set the global variable[symbol] &MID to an EBCDIC character string so that, when the variable[symbol] is coded as the first portion of the message text field of an operating system WTO macro instruction, the correct message identification is displayed with the message.

This macro instruction should be coded directly before the WTO macro instruction.

### **Format description**

➤ ┌──────────┐ \$MID — *id-value* ➤  
└── *symbol* ─┘

#### **id-value**

Specifies the numeric 3-digit message identification of the message appearing in the succeeding WTO macro expansion.

**Note:** Coding should not depend on the exact length or format of the character string assigned to the &MID variable symbol.

### **Environment**

- Main task, subtask, or user address space.
- \$WAIT cannot occur.

## **\$MODCHK – Load module verification**

---

Use the \$MODCHK macro instruction to request that JES2 verify the named load module routine. \$MODCHK can verify:

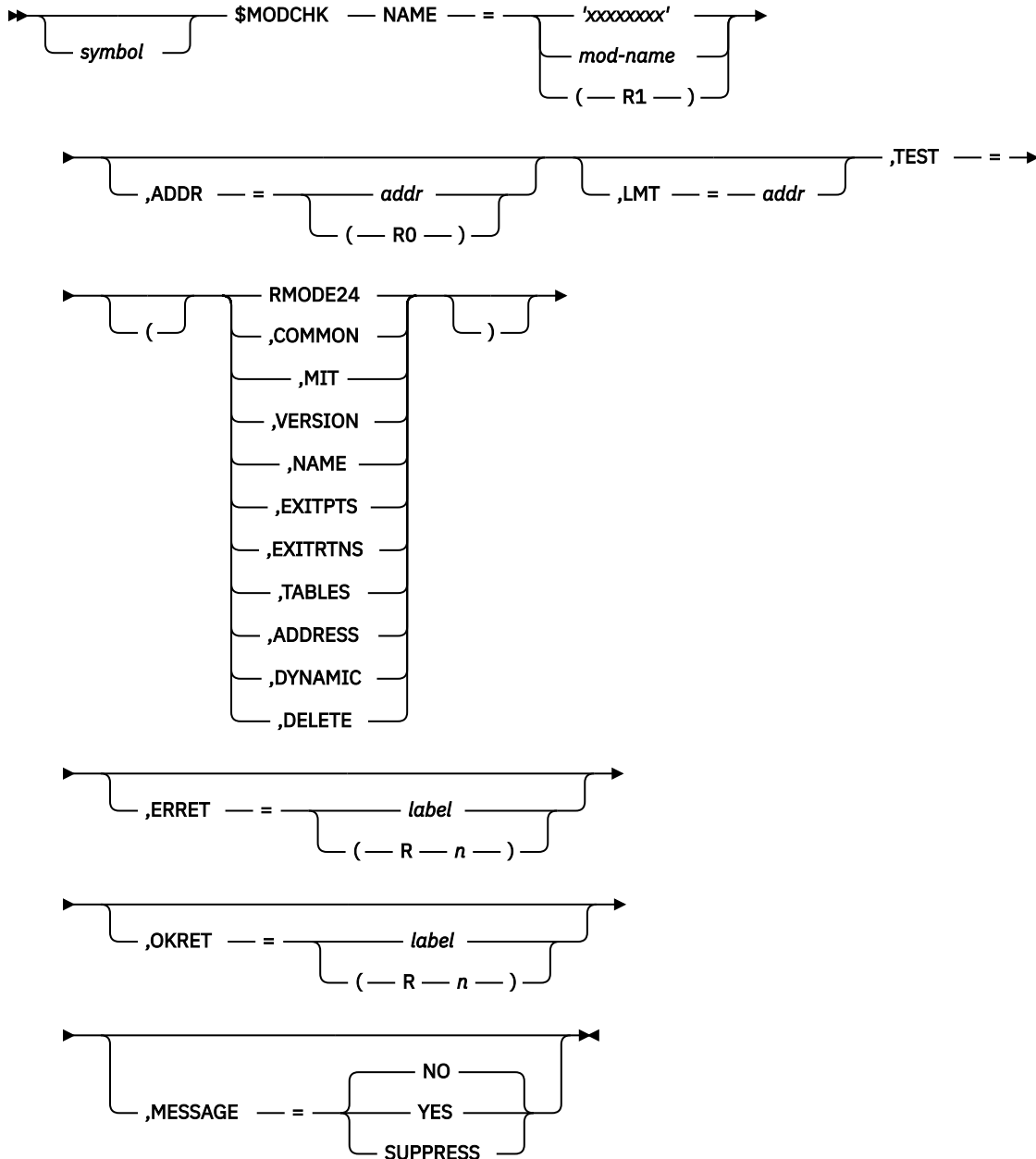
- if the module resides below 16 megabytes in virtual storage
- if the module resides in common storage
- if the module was assembled at the same version as the JES2 nucleus and with the correct level of macros
- if the module name matches that specified in the MIT (MITNAME)

In addition, \$MODCHK can:

- propagate, to the XIT, \$EXIT points that are defined in the module
- resolve, from the module's \$ENTRY points, \$EXIT routines

Specifically, this macro is useful to guarantee that you have not inadvertently attempted to mix MVS versions and that all modules are assembled at the same system product (SP) level. This early verification and notification prevents an attempt by JES2 to load an incorrect module and eventually terminate. An unsuccessful verification causes JES2 to issue message \$HASP875 with a specific reason text.

## Format description



### NAME=

Specifies the name of the load module or assembly module to be verified by \$MODCHK. Specify a 1- to 8-character module name, a label referencing the beginning of the module, or a register (R1-R10) containing the address of the module. This is a required keyword.

### ADDR=

Specifies the address of this module by either a label or a register (R0, R2-R10) containing the module address. You must code this keyword if this module is not loaded by a \$MODLOAD or the module was loaded by \$MODLOAD and TYPE=OS was specified. Otherwise, this keyword is optional; if it is not coded, the address is taken from the \$LMT entry that JES2 built when the module was loaded by \$MODLOAD.

### LMT=

Specifies the address of the LMT that contains the module. The address of the module is passed in ADDR=.

**TEST=**

Specifies which module verification tests are to be performed. If you specify more than one test type, enclose the list in parentheses and separate each type by a comma, for example, TEST=(NAME,RMODE24,VERSION). This is a required keyword.

<b>Test Type</b>	
<b>Meaning</b>	

**RMODE24**

Tests that the module resides below 16-megabytes in virtual storage.

**COMMON**

Tests that the module resides in common storage.

**MIT**

Tests that the module is large enough to contain the MIT, that the MIT entry table pointer points to a valid field within the module, and that the MIT is located at the beginning of the module.

**VERSION**

Tests that the version of JES2 and this module are at the same level and that all macros contained in the module are assembled at the correct level of JES2.

**NAME**

Tests that the NAME= keyword specifies the same name as the MITNAME specified in the MIT.

**EXITPTS**

Propagates, to the XIT, any \$EXIT points that are defined in the module.

**EXITRTNS**

Resolves any module \$ENTRY points with \$EXIT routines.

**TABLES**

Resolves any dynamic tables within the module.

**ADDRESS**

Resolves routine addresses.

**DYNAMIC**

Tests that the module supports the LOADMOD command.

**DELETE**

Tests that the module can be deleted.

**ERRET=**

Specifies the address of an error routine that is to get control if the module fails the test (that is, R15 is nonzero). Specify the address by either a label or a register containing the address of this error routine. An error message can be returned if MESSAGE=YES is also coded.

**OKRET=**

Specifies the address of a routine that is to get control if the module passes the test (that is, R15 is zero). Specify the address by either a label or a register containing the address of this routine.

**MESSAGE=**

Specifies whether (YES) or not (NO) JES2 issues message \$HASP875 if any test fails.

<b>Message Type</b>	
<b>Meaning</b>	

**YES**

If any errors are encountered, JES2 issues the \$HASP875 message.

**NO**

If any errors are encountered, JES2 does not issue the \$HASP875 message (returns message in R1). The caller must free the 128-byte-message work area returned (Subpool = 0 key = 1).

**SUPPRESS**

Does not issue or return error message.

## Return codes

The following return codes (in decimal) are returned in register 15.

### Return Code Meaning

**0**

Tests were successful.

**4**

Tests failed. \$MODCHK also returns a reason code that provides additional information about why the tests failed.

## Reason codes for return code 4

When \$MODCHK returns control, register 1 points to a storage area that contains message text and a reason code. This storage is structured as described below (byte 1 refers to the byte that's pointed to by register 1; byte *n* refers to the last byte of the storage area).

### Byte position Contents

#### byte 1 and 2

These bytes are not part of the intended programming interface.

#### byte 3

The length of the message text portion of the variable length text field that follows.

#### byte 4 to byte *n*

Variable length text field. This field contains message text that explains the reason code, followed by a comma and a blank character. The blank character is followed by the character string RC=*nn*. The variable *nn* contains one of the following reason codes in character format. For a detailed explanation of the reason codes, see documentation for JES2 message \$HASP875 in [z/OS JES2 Messages](#).

### Reason Code Meaning

**06**

The RMODE must be 24.

**07**

The module name does not match its MIT.

**08**

The version of the module does not match the version of JES2.

**09**

The module and JES2 were assembled with different levels of MVS macros.

**10**

The module is not in common storage.

**11**

The module's MIT is not valid.

**12**

JES2 could not find the module.

**15**

The module user version character string does not match the version of JES2 that's running.

**16**

The module IBM user version is not valid.

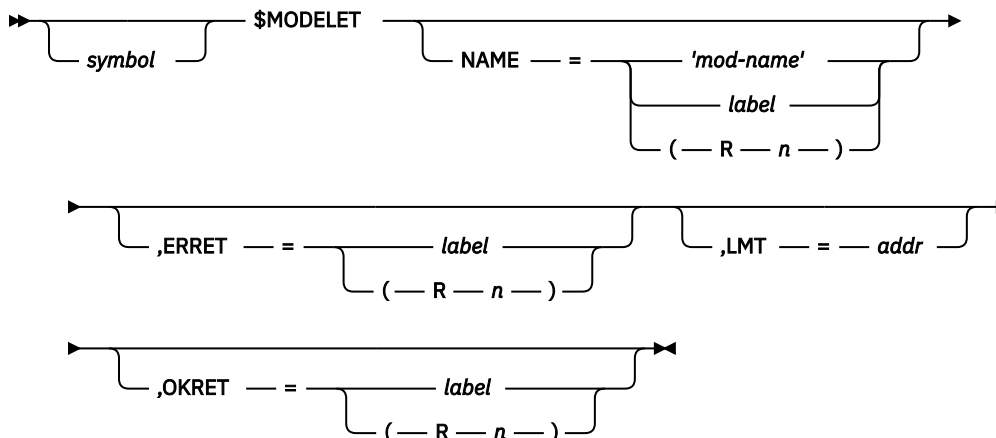
## Environment

- JES2 main task limited (initialization).
- MVS WAIT may occur.

## \$MODELET – Delete a load module

Use the \$MODELET macro to delete a specified JES2 load module and also invalidate the load module table (\$LMT) entry for this module. You can use this macro instruction only if the module was loaded by a \$MODLOAD and TYPE=JES2 was specified.

### Format description



#### NAME=

Specifies the name of the module to be deleted. Specify a 1- to 8-character module name, a label referencing the beginning of the module, or a register (R1-R10) containing the address of the name. This is a required keyword.

#### ,ERRET=

Specifies the label or register that contains the address of an error routine that receives control if the module is not successfully deleted.

#### ,LMT=

Specifies the address of the LMT to delete. You can use LMT= to delete a specific instance of a load module.

#### ,OKRET=

Defines a location (label or register) to branch if R15 is zero on return from \$MODELET.

### Return codes

The following return codes (in decimal) are returned in register 15.

#### Return Code

##### Meaning

0

Load module successfully deleted.

4

Module not loaded by \$MODLOAD with TYPE=JES2.

### Environment

- JES2 main task and initialization.
- no MVS or JES2 waits can occur.



#### Attention:

The caller is responsible for ensuring that exit pointers into the module can be changed to zero at this time. It is recommended that you use this macro only when all exits are disabled or during JES2 initialization.

## \$MODEND – Generate end of module

Use \$MODEND to generate the MIT entry table (MITETBL) to fill in the 256-bit mask field in the MIT according to what exits are defined within the module, and to calculate the length for the CSECT created by \$MODULE.



**Attention:** This macro instruction must be coded at the end of every module, with no exceptions.

### Format description

➤ symbol \$MODEND ➤

### Environment

- All environments.
- \$WAIT is not applicable.

## \$MODLOAD – Load module load

Use \$MODLOAD to call the \$MODLOAD routine that loads JES2 modules (supplied by IBM and installation) and MVS style modules. The caller must perform any needed \$MODCHKs on the module that was loaded, including resolving any dynamic tables or exit routines.

**Note:** If \$MODLOAD is used for a module that has already been loaded and not deleted with \$MODELET, the processing proceeds based on the REPLACE= value. If REPLACE=NO (the default), the module is not reloaded and the existing module returns. The subpool value is ignored. If REPLACE=YES is specified and the module is loaded, a new copy of the module is loaded and a new LMT is built. You must delete the old module and LMT using \$MODELET.

### Format description

➤ symbol \$MODLOAD

NAME = 'mod-name'

label

(— R — n —)

,TYPE = JES2

OS

,SUBPOOL = nnn

LPA

,MESSAGE = NO

IFEXISTS

YES

SUPPRESS

,ERRET = label

(— R — n —)

,REPLACE = NO

YES

➤

## **\$MODLOAD**

### **NAME=**

Specifies the name of the module to be loaded by MODLOAD. Specify a 1- to 8-character module name, a label of a field that contains the name, or a register that contains the address of the name. This is a required keyword.

### **SUBPOOL=**

Specifies the storage location into which the module should be loaded.

#### ***nnn***

Specifies the subpool (0 to 255) for the directed load. The module is to be loaded using the standard module search algorithms.

### **LPA**

Specifies the module is in LPA.

### **Note:**

1. This parameter is supported for OS load modules only if SUBPOOL=LPA is specified.
2. It is suggested this parameter be used to load into the common storage area. However, all loads are supported.

### **TYPE=**

Specifies the type of load module to be loaded. If the specified module is a JES2 module and successfully loaded, it receives a \$LMT entry; OS load modules do not.

### **MESSAGE=**

Specifies whether \$MODLOAD issues the \$HASP875 message if the load fails (return code 8).

#### **IFEXISTS**

If JES2 finds the module and the module has errors, \$MODLOAD issues the message. If JES2 is unable to find the module, \$MODLOAD does not issue the message.

#### **YES**

If any errors are encountered, \$MODLOAD issues the message.

#### **NO**

If any errors are encountered, \$MODLOAD does not issue the HASP875 message (returns message in R1). The caller must free the 128 byte message work area returned (Subpool = 0 key = 1).

#### **SUPPRESS**

Does not issue or return error message.

### **ERRET=**

Specifies the label or register that contains the address of an error routine that receives control if the module is not successfully loaded.

### **REPLACE=**

Specifies whether \$MODLOAD fails a request or replaces the existing module if the module is already loaded.

#### **NO**

If the module is already loaded, \$MODLOAD fails a request.

#### **YES**

If the module is already loaded, \$MODLOAD replaces it. If the module is not loaded, \$MODLOAD does not load it.

**Note:** REPLACE=YES does not delete the old module. The caller must perform \$MODCHKs on the new module. If everything passes, call \$MODELET with the old LMT to delete the original module.

## **Return codes**

The following return codes (in decimal) are returned in register 15.

<b>Return Code</b>	<b>Meaning</b>
--------------------	----------------



**0**

Load module is successfully loaded. If TYPE=JES2, register 1 contains the address of the \$LMT. If TYPE=OS, register 1 contains the address of the load module.

**4**

If REPLACE=NO is specified, the load is not done because the module has already been loaded. R1 contains the LMT entry address. The information of load location requested on this call is ignored.

If REPLACE=YES is specified, the load is not done because the module is not loaded or is not loaded in the specified load location. R1 contains the LMT entry address or a value of zero.

**8**

Load fails. If MESSAGE=NO, R1 points to a work area that contains AL1(DIAGLEN),C'DIAGNOSTIC'.

## Reason codes for return code 8

When \$MODLOAD returns control, register 1 points to a storage area that contains the message text and a reason code. The structure of this storage is as follows: (byte 1 refers to the byte that register 1 points to; byte n refers to the last byte of the storage area)

### Byte position

#### Contents

#### byte 1 and 2

These bytes are not part of the intended programming interface.

#### byte 3

The length of the message text portion of the variable length text field that follows.

#### byte 4 to byte n

Variable length text field. This field contains message text that explains the reason code, followed by a comma and a blank character. The blank character is followed by the character string RC=*nn*. The variable *nn* contains one of the following reason codes in character format. For a detailed explanation of the reason codes, see documentation for JES2 message \$HASP875 in [z/OS JES2 Messages](#).

#### Reason Code

#### Meaning

**01**

There is insufficient storage for the load module table entry.

**02**

The module cannot be loaded.

**03**

The module is not reentrant.

**04**

New CSA or LPA are not valid on a hot start.

**05**

There is insufficient storage to load the module

**14**

An exit 0 routine made a \$MODLOAD call.

## Environment

- JES2 main task and initialization.
- MVS WAIT may occur.



### Attention:

An exit 0 routine must not use \$MODLOAD to load common storage JES2 modules.

## **\$MODULE – Prepare a JES2 module or expand control block mappings**

---

You must use \$MODULE to prepare a JES2 exit module or any other JES2 module to run in one of the JES environments. You can also use \$MODULE in any other module to expand mappings of certain MVS or JES2 control blocks.

### **Preparing a JES2 module**

You must code \$MODULE once in each module immediately after a COPY \$HASPGBL assembler statement and before any other code. **No JES2 modules are exceptions to this rule.** \$MODULE allows you to:

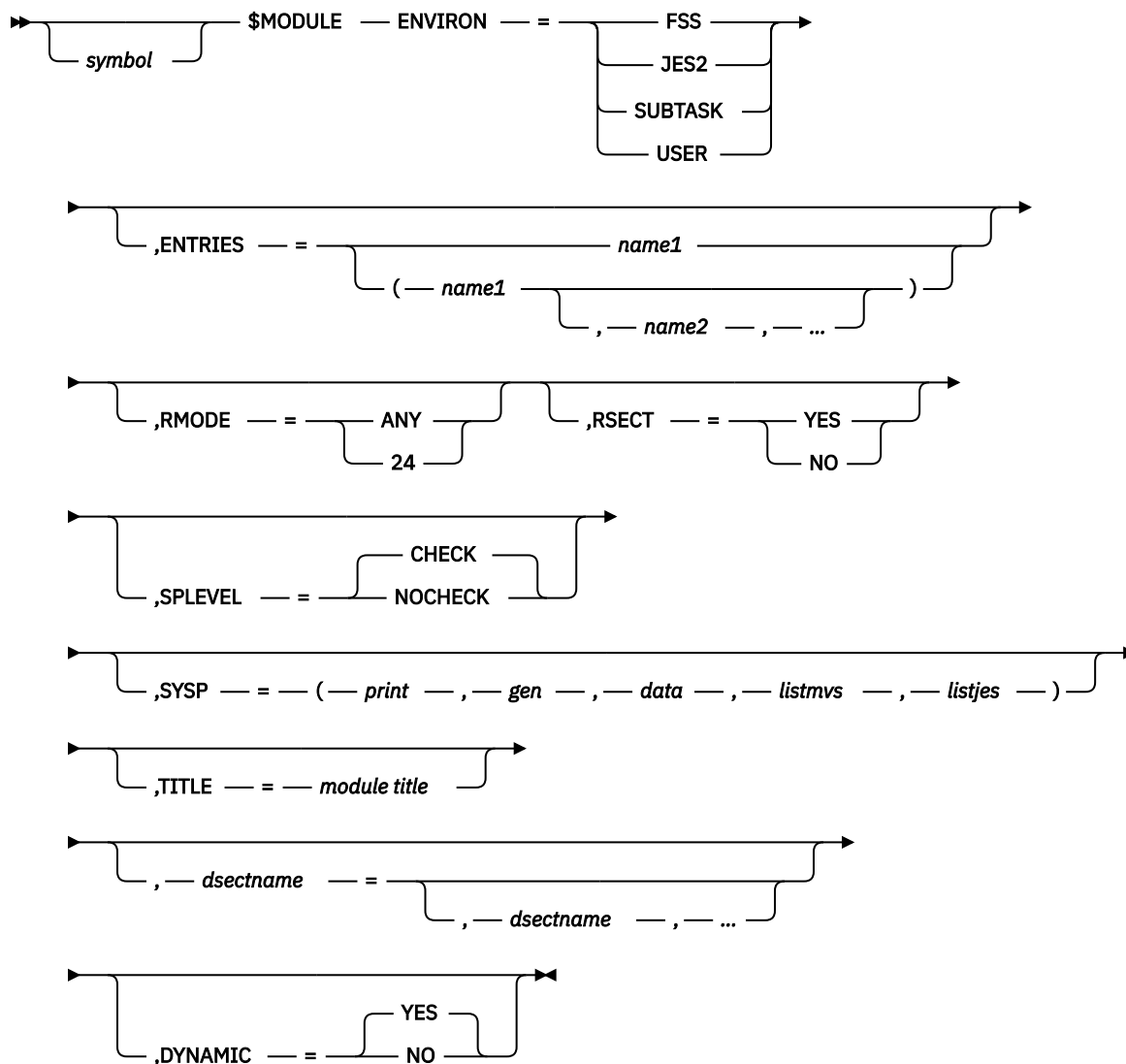
- Name the module and define the JES environment in which it runs.
- Establish the module's RMODE.
- Define the module as either read-only (reentrant) or non-read-only.
- Request that JES2 verify whether the module was assembled using the correct level of the MVS macro library.
- Specify JES2 or MVS control block mappings that \$MODULE is to include in the module. The table at the end of this topic lists the control block mappings (DSECTs) that you can specify.
- Title the assembly listing and control printing of that listing.

### **Expanding MVS or JES2 control block mappings**

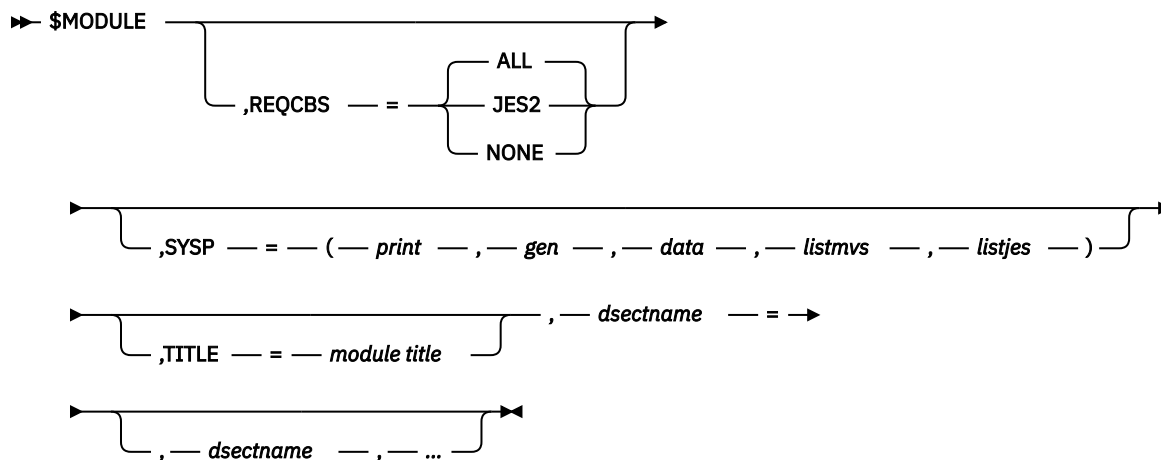
You can code \$MODULE in any module, such as an SMF exit, to include control block mappings in the module. \$MODULE allows you to:

- Specify JES2 or MVS control block mappings that \$MODULE is to include in your module. The table at the end of this topic lists the control block mappings (DSECTs) that you can specify.
- Request that \$MODULE includes in your module implicitly required control block mappings.
- Title the assembly listing and control the printing of that listing.

## Format description - Preparing a JES2 module



## Format description - Expanding control block mappings



## Parameter descriptions

**Note:** The parameters that you can use depends upon which form of the \$MODULE macro you use. To determine which parameters you can use, see either [“Format description - Preparing a JES2 module”](#) on page 245 or [“Format description - Expanding control block mappings”](#) on page 245.

### **symbol**

Specifies the name that you want assigned to the control section that you are defining. \$MODULE generates a CSECT or RSECT statement and assigns this name to that statement. \$MODULE also assigns this name to the assembly variable &J2SECTN and the string CSECT or RSECT to the assembly variable &J2SECTT.

**Note:** Do not specify this name on any other \$ENTRY or \$MODULE statements.

### **ENVIRON=**

Specifies the environment in which this module runs. \$MODULE sets the assembly variable &ANVIRON equal to the value you specify on the ENVIRON= parameter.

### **FSS**

The module runs in the functional subsystem environment. If you omit the RSECT= parameter, \$MODULE generates an RSECT statement and assigns the string RSECT to the assembly variable &J2SECTT. \$MODULE establishes addressability to the HFCT by generating the following USING statement in the assembly module:

```
USING HFCT,R11
```

### **JES2**

The module runs in the JES2 main task environment. If you omit the RSECT= parameter, \$MODULE generates a CSECT statement and assigns the string CSECT to the assembly variable &J2SECTT. \$MODULE establishes addressability to the HCT and the PCE by generating the following USING statements in the assembly module:

```
USING HCT,R11  
USING PCE,R13
```

### **SUBTASK**

The module runs in the JES2 subtask environment. If you omit the RSECT= parameter, \$MODULE generates an RSECT statement and assigns the string RSECT to the assembly variable &J2SECTT. \$MODULE establishes addressability to the HCT by generating the following USING statement in the assembly module:

```
USING HCT,R11
```

### **USER**

The module runs in the user environment. If you omit the RSECT= parameter, \$MODULE generates an RSECT statement and assigns the string RSECT to the assembly variable &J2SECTT. \$MODULE establishes addressability to the HCCT by generating the following USING statement in the assembly module:

```
USING HCCT,R11
```

### **ENTRIES=**

Specifies one or more names of tables or routines in the assembly module. \$MODULE creates entries for these names in the module-end table, thus treating these names as though they were defined on the \$ENTRY macro.

If your program builds tables by using any of the following macros, use the ENTRIES= parameter to create entries in the module-end table:

- \$DCTTAB
- \$DTETAB
- \$PCETAB

- \$RDIDTAB
- \$SCANTAB
- \$SYMTAB
- \$TIDTAB
- \$VERTAB
- \$WSTAB

**REQCBS=**

Specifies whether \$MODULE is to generate:

- Control block mappings that are implicitly required by the control block mappings that are specified on the \$MODULE macro.
- Control block mappings that are required by the particular assembly environment.

**ALL**

Generate mappings of all implicitly required JES2 and MVS control blocks and mappings of all control blocks required by the assembly environment.

**JES2**

Generate mappings of only implicitly required JES2 control blocks and mappings of JES2 control block mappings that are required by the assembly environment.

**NONE**

Do not generate mappings of any implicitly required control blocks or mappings of any control blocks required by the assembly environment.

**RMODE=**

Specifies the control section's residence mode. If you want \$MODULE to generate an RMODE assembler statement within the control section, code the RMODE= parameter. If you do not want an RMODE assembler statement that is generated, omit this parameter.

**ANY**

The control section can be placed above or below 16 megabytes.

**24**

The control section must be placed below 16 megabytes.

**RSECT=**

Specifies whether the control section is read-only (reentrant).

**YES**

The control section is read-only. \$MODULE generates an assembler RSECT statement and assigns the string RSECT to the assembly variable &J2SECTT. The assembler also performs some checks to determine whether the control section violates any rules of reentrant programming.

**NO**

The control section is not read-only. \$MODULE generates an assembler CSECT statement and assigns the string CSECT to the assembly variable &J2SECTT.

If you omit this parameter, \$MODULE uses the value that is specified on the ENVIRON= parameter to determine whether to generate a CSECT or an RSECT statement.

**SPLEVEL=**

Specifies whether JES2 is to check each installation provided module that it loads to ensure that the module was assembled with the correct level of the MVS macro library.

**CHECK**

JES2 is to perform the check. This is the default and the option that IBM suggests you select.

**NOCHECK**

JES2 is to bypass checking. IBM **strongly** recommends that you specify SPLEVEL=CHECK.

**SYSP=(print,gen,data,listmvs,listjes)**

Specifies values that control the printing of the assembly listing.

**print**

Controls whether code generated by the \$SCANTAB macro is printed in the assembly listing.

- To suppress printing the \$SCANTAB macro statements and the generated code, specify OFF.
- To suppress printing the generated code, specify NOGEN.
- If you want to continue using the PRINT values that are in use at the time you issue \$SCANTAB, specify ON or GEN.

The default is NOGEN.

**Note:** \$MODULE assigns the print value to the assembly variable &J2PRTSW. Installations can use this variable to control printing of installation-provided macros that are used within this module.

**gen**

Controls whether expansions of executable macros are printed in the assembly listing.

- To print macro expansions, specify GEN.
- To suppress printing of macro expansions, specify NOGEN.

The default is GEN.

**data**

Controls printing of data constants in the assembly listing.

- To print all of the object code generated for each constant, specify DATA.
- To suppress printing of all but the first eight bytes of object code generated for each constant, specify NODATA.

The default is NODATA.

**Note:** If you have specified NOGEN for the gen positional parameter, the data parameter has no effect on constants generated during macro processing.

**listmvs**

Controls whether MVS DSECTs that are specified on the \$MODULE macro are printed in the assembly listing.

- To print the DSECTs, specify GEN.
- To suppress printing the DSECTs, specify NOGEN.

The default is NOGEN.

**listjes**

Controls whether JES2 DSECTs that are specified on the \$MODULE macro are printed in the assembly listing.

- To print the DSECTs, specify GEN.
- To suppress printing the DSECTs, specify NOGEN.

The default is NOGEN.

**Note:** If you are coding the module definition form of \$MODULE, at the time you assemble your module you can override values coded on SYSP= by using the assembler variable &SYSPARM as follows:

```
PARM='SYSPARM(option-1[,option-2,...option-n])'
```

Option-1, option-2, and so forth, correspond to the positional parameters on the SYSP parameter.

**TITLE=**

Specifies a character string title for this module.

**dsectname**

Identifies the MVS and JES2 dsect mappings that are to be included in this control section. You can specify the dsect names in any order. The order in which you specify the dsect names has no effect on the order in which they appear in your assembled module.

Each dsectname can be specified as one of the DSECTIDs shown in the following table or as (dsectid,genid). The variable genID can either be specified as GEN or NOGEN and overrides the “listmvs” and “listjes” values (for the particular macro) specified through the SYSP= parameter.

**DYNAMIC=**

Defines whether this module supports the \$ADD, \$TREFRESH, and \$DEL LOADMOD commands.

This parameter is optional for the module-defining case. Do not use it for the mapping-only case.

**YES**

This module supports dynamic load module commands.

**NO**

This module can only be loaded or deleted by initialization statements.

**Note:** For modules that are loaded by initialization statement or commands, this parameter can only be checked in the first CSECT in the module. JES2 does not process subsequent CSECTs, so the subsequent CSECTs cannot affect how the load module is processed.

[Table 7 on page 250](#) lists all the MVS DSECTs that can be specified on the \$MODULE macro. [Table 8 on page 251](#) lists all the JES2 DSECTs that can be specified on the \$MODULE macro.

Table 7. MVS DSECTIDs That Can Be Specified on \$MODULE

DSECTID	DSECTID	DSECTID	DSECTID
ACB	ENF78	KEYS	SDRMT
ACBXL	ENF83	LAA	SDUMP
ACEE	ENFP	LCA	SDWA
ABDPL	ENFSG	LCT	SIOT
ADSR	ENOBJ	LDA	SJPUP
AJCTB	EPVT	LGDAT	SJRC
ARA	ESSY	LGINF	SJREP
ARL	ETD	LGSTP	SJRSP
ASCB	EWA	LIMD	SJRUP
ASEO	EXAA	LLE	SJSCP
ASSB	EXRET	LPDE	SJSMP
ASVT	EZAENF80	LPRET	SJSYD
ASXB	EZASMI	LWA	SJTRC
ATB	FDF	MOND	SJTRP
BASEA	FRRS	MGCR	SJTSP
BIND	FSCT	MGCRE	SMCA
BPXYCONS	FSIP	MLTE	SMF30
BPXYENFO	FSVT	NAMP	SPLIO
BPXYIOCC	GDA	NCC	SPP
BPXYOEXT	GEPL	NEL	SRB
BPXYSOCK	HWTH	NEPL	SRMENF1
BTOKP	HWTJ	NIB	SSCT
CDE	IAZYTDBC	NTASM	SSIB
CDR	IAZYTNSM	NTKP	SSJD
CIDF	IAZYTNRQ	OHLD	SSJM
CKPD	IAZYTPRM	ORE	SSJP
CMPL	IAZYTCT	OUCB	SSL
CNMB	IAZYTSTCT	PATH	SSOB
CNPRM	IAZYTTRC	PDS	SSPJ
CON	ICYENF	PJCO	SSSF
CONA	IDX	PMAP	SSST
CONV	IHATEDS	PPL	SSS2
CSCB	IEDB	PRC	SSTS
CSRC4ASM	IOB	PSA	SSVT
CSVMODI	IOBE	PSCB	STCB
CTE	IOCM	PSL	SVTX
CTOKEN	IOSB	PSW	SYMBP
CTRACE	IOSCAPU	PVT	SYMDF
CTXI	IRABQS	QMIDS	TCB
CVDEV	ISGE51CN	QMPA	TCT
CVT	ISGRIB	QUAA	TEXT
CWPL	ISGYCON	RASD	TIOT
DCB	ITRLP	RAX	TOKEN
DCBE	ITSPP	RB	TRKCALC
DCBXL	IWMSVPOL	RESPA	TTBF
DEB	IWMYCON	RMPL	TTBL
DECB	IXCYENF	RMR	UCB
DGSP	JBCLD	RPL	UCM
DOTUM	JCOR	RQE	UPFX
DSAB	JESCT	RXTW	VREC
DSCB	JFCB	SAFP	VRL
DSERV	JFCB	SAV	WLMENF1
DSINF	JICA	SCB	WLMENF56
DVA	JMRE	SCT	WLMENF57
DYN	JPCKP	SDMSE	WPL
DYNM	JPCLS	SJACP	WQE
EAECB	JPITD	SJDLP	XSB
ECB	JPLEX	SJEXP	XSSP
ECVT	JPLXI	SJERP	XTLST
EEPL	JPNJN	SJFNP	YIXAC
ENF58	JPROC	SJGEP	YIXEN
ENFCT	JPSPL	SJKEY	YIXIF
ENFPM	JSAB	SJKLP	YIXJE
ENFSG	JSCB	SJMRP	YIXPE
ENF40	JSPA	SJOKY	YIXSE
ENF70	JSQRY	SJPRFX	YSUDF
	JSRC		Z\$XPL



Table 8. JES2 DSECTIDs That Can Be Specified on \$MODULE

DSECTID	Macros	Description of Code Generated
\$ACT	\$ACT	Automatic command table DSECT
\$ALINDEX	\$ALIN	ALET index table DSECT
\$ALIWORK	\$ALIWORK	ALICE PCE work area
\$APT	\$APT	NJE/SNA application table DSECT
\$ARMG	\$ARMG	ARM support JESXCF message DSECT
\$ARMT	\$ARMT	ARM support trace record
\$ARMWORK	\$ARMW	ARM processor PCE work area DSECT
\$ARTABL	\$ARTABL	Allocated Resource Table DSECT
\$ASYWORK	\$ASYW	Asynchronous I/O PCE Work Area
\$AUXCB	\$AUXCB	Auxiliary Address Space Control Block
\$BAT	\$BAT	Buffer AUXILIARY table DSECT
\$BFW	\$BFW	3800 buffer work area DSECT
\$BLDMSG	\$BLDM	Build message parameter list DSECT
\$BTG	\$BTG	BADTRACK group element DSECT
\$BUFFER	\$BUFFE	I/O buffer DSECT
\$CADDR	\$CADDR	Common storage address table DSECT
\$CAL	\$CAL	Change LOG address list DSECT
\$CALE	\$CALE	Change LOG address list element
\$CAPE	\$CAPE	Communications access parameter element
\$CAT	\$CAT	Class attribute table DSECT
\$CBP	\$CBP	CBIO work area
\$CCE	\$CCE	Cell control element DSECT
\$CCW	\$CCW	Channel command word definitions
\$CDCT	\$CDCT	Common Device Control Table
\$CDCTQS	\$CDCTQS	Common Device Queue Heads
\$CDCWORK	\$CDCWORK	CDC Processor PCE work area
\$CDI	\$CDI	Configuration directory
\$CHK	\$CHK	(MVS) FSI checkpoint record DSECT
\$CICB	\$CICB	C/I Address Space Block
\$CID	\$CID	Connect ID cell
\$CIPARM	\$CIPARM	C/I Parameter List
\$CIRWORK	\$CIRW	Common initialization routine PCE work area DSECT
\$CIWORK	\$CIWORK	C/I Work Areas
\$CK	\$CKLI	Checkpoint block DSECT

*Table 8. JES2 DSECTIDs That Can Be Specified on \$MODULE (continued)*

<b>DSECTID</b>	<b>Macros</b>	<b>Description of Code Generated</b>
\$CKGPAR	\$CKGPA	Checkpoint generalized parameter area
\$CKM	\$CKM	Checkpoint inter-member communication area
\$CKPINFO	\$CKPINFO	Checkpoint information
\$CKPRECV	\$CKPR	Checkpoint recovery dialog work area
\$CKPTQCB	\$CKPT	Checkpoint request control block
\$CKPWORK	\$CKPW	Checkpoint processor PCE work area DSECT
\$CKV	\$CKV	Checkpoint verification table
\$CKW	\$CKW	Checkpoint work area
\$CKX	\$CKX	Checkpoint reconfiguration JESXCF messages
\$CLASSGRP	\$CLASSGRP	Class Group DSECT
\$CMB	\$CMB	Console message buffer DSECT
\$CNVWORK	\$CNVW	Conversion processor PCE work area DSECT
\$COMWORK	\$COMW	Command processor PCE work area DSECT
\$CPCWORK	\$CPCW	Cell pool query cell work area
\$CPEBE	\$CPEBE	Cell pool extent block element
\$CPINDEX	\$CPIN	Cell pool index table
\$CPMASTR	\$CPMA	Cell pool master table
\$CPPWORK	\$CPPW	Cell pool query pool work area
\$CPXWORK	\$CPXW	Cell pool query extent work area
\$CPT	\$CPT	Compaction table DSECT
\$CRB	\$CRB	Checkpoint/restart buffer area DSECT
\$CRE	\$CRE	Command redirection element DSECT
\$CRTSYN	\$CRTSYN	Create SYSIN Data Set Parameter List
\$CSVPARM	\$CSVPARM	CSV \$\$\$\$LOAD/\$\$\$\$DEL Parm List
\$CTKNENF	\$CTKNENF	Service parameter list
\$CTW	\$CTW	Checkpoint trace work area DSECT
\$CWA	\$CWA	MCS console work area DSECT
\$DAIR	\$DAIR	DAIRFAIL parameter list DSECT
\$DAS	\$DAS	Direct access spool data set DSECT
\$DAWNWRK	\$DAWNWRK	DAWN PCE work area
\$DCHKWK	\$DCHKW	DESTCHK authorization work area DSECT
\$DCT	\$DCT	Device control table DSECT
\$DCTTAB	\$DCTTA	DCT table (\$GETABLE) DSECT
\$DILWORK	\$DILWORK	Do it later PCE work area

*Table 8. JES2 DSECTIDs That Can Be Specified on \$MODULE (continued)*

<b>DSECTID</b>	<b>Macros</b>	<b>Description of Code Generated</b>
\$DJB	\$DJB	Duplicate job block
\$DLS	\$DLS	Deadline Scheduling
\$DLSWORK	\$DLSWORK	Deadline Scheduling PCE work area
\$DSB	\$DSB	Data space block DSECT
\$DSCA	\$DSCA	Data set catalog DSECT
\$DSCT	\$DSCT	Data set control table DSECT
\$DSSCB	\$DSSCB	Data set services control block DSECT
\$DSET	\$DSET	SPOOL data set information
\$DSIX	\$DSIX	Data Set Index DSECT
\$DSTA	\$DSTA	Userdest work area DSECT
\$DSWA	\$DSWA	Data space services work area
\$DTE	\$DTE	Daughter task element DSECT
\$DTEACCT	\$DTEACCT	Account DTE work area extension DSECT
\$DTEALOC	\$DTEALOC	DYNALLOC DTE work area extension DSECT
\$DTEASST	\$DTEASST	SPOOL Migrator Assistant work area
\$DTECKCF	\$DTECKCF	CKPT on CF DTE work area extension DSECT
\$DTECKDA	\$DTECKDA	CKPT on DASD DTE work area extension DSECT
\$DTECKVR	\$DTECKVR	CKPT VERS DTE work area extension DSECT
\$DTECNV	\$DTECNV	Conversion DTE work area extension DSECT
\$DTEIMG	\$DTEIMG	IMAGE DTE work area extension DSECT
\$DTELIM	\$DTELIM	Resource Limits DTE work area DSECT
\$DTEMIGR	\$DTEMIGR	SPOOL Migrator DTE work area
\$DTEOFF	\$DTEOFF	Offload DTE work area extension DSECT
\$DTESPL	\$DTESPL	Spool DTE work area extension DSECT
\$DTESUBS	\$DTESUBS	General subtask work area extension DSECT
\$DTETAB	\$DTETAB	DTE table (\$GETABLE) DSECT
\$DTEVTAM	\$DTEVTAM	VTAM DTE work area extension DSECT
\$DTEWTO	\$DTEWTO	WTO DTE work area extension DSECT
\$EDS	\$EDS	Email Delivery Services
\$EDSWORK	\$EDSWORK	EDS PCE Work Area
\$ENFWORK	\$ENFWORK	HASP ENF LISTEN PCE Work Area
\$ERA	\$ERA	Error recovery area DSECT
\$ERPL	\$ERPL	\$Error parameter list DSECT
\$ERRTAB	\$ERRTAB	Error Count Table DSECT

*Table 8. JES2 DSECTIDs That Can Be Specified on \$MODULE (continued)*

<b>DSECTID</b>	<b>Macros</b>	<b>Description of Code Generated</b>
\$EST	\$EST	Estimated counts DSECT
\$EVT	\$EVT	ENF Listen Event DSECT
\$EXTENDS	\$EXTENDS	Extend Data Set DSECT
\$EZA	\$EZA	EZASMI work areas
\$FCLWORK	\$FCLWORK	FSS cleanup on EOM PCE work area
\$FMH	\$FMH	SNA function management header DSECT
\$FMTADJ	\$FMTADJ	IPCS value adjustment DSECT
\$FMTCTAB	\$FMTCTAB	Control block format table DSECT
\$FRDR	\$FRDR	File Reader Work Area
\$FSACB	\$FSAXB	Functional subsystem application extension DSECT
\$FSAXB	\$FSAXB	FSA Control Block Extension
\$FSIEQU	\$FSIEQU	FSI equates
\$FSSCB	\$FSSWORK	HASP FSS control block DSECT
\$FSSWORK	\$FSSWORK	HASP FSS Support PCE Work Area
\$FSSXB	\$FSSXB	Functional subsystem control block extension DSECT
\$GASSIGN	\$GASSIGN	Assign grouping token parameter list DSECT
\$GCB	\$GCB	GETREC chain control block DSECT
\$GGEQU	\$GGEQU	Generic grouping equates
\$GKGET	\$GKGET	GET grouping keys parameter list DSECT
\$GKINIT	\$GKINIT	Initialize grouping keys parameter list DSECT
\$GPQE	\$GPQE	General purpose subtask queue element
\$GRPKWD	\$GRPKWD	Output processor grouping keywords
\$GRPLIST	\$GRPLIST	Output grouping parameter list
\$GSINIT	\$GSINIT	Initialize grouping strings parameter list DSECT
\$GSTERM	\$GSTERM	Terminate grouping strings parameter list DSECT
\$GTW	\$GTW	\$#GET trace work area DSECT
\$HASB	\$HASB	Address space block DSECT
\$HASXB	\$HASXB	Address space extension block DSECT
\$HCCT	\$HCCT	Common storage communication table
\$HCT	\$HCT	HASP control table
\$HDP	\$HDP	Control block pool header DSECT
\$HFAM	\$HFAM	File allocation map
\$HFAME	\$HFAME	File allocation map entry
\$HFCT	\$HFCT	FSS communications table

Table 8. JES2 DSECTIDs That Can Be Specified on \$MODULE (continued)

<b>DSECTID</b>	<b>Macros</b>	<b>Description of Code Generated</b>
\$HJCT	\$HJCT	Monitor Communication Table
\$HSU	\$HSU	HOCSETUP parameter list
\$ICE	\$ICE	SNA interface control element DSECT
\$IEW	\$IEW	IOT I/O error recovery work area
\$IFMTABL	\$IFMTABL	IPCS format table GEN and DSECT
\$INIWARM	\$INIWARM	HASPIR* to warmstart communications table
\$IOT	\$IOT	Input/output table DSECT
\$IOTERR	\$IOTERR	Spin IOT error recovery
\$IRCWORK	\$IRCWORK	INTRDR Cleanup PCE work area
\$IRE	\$IRE	Internal reader tracking element
\$IRIS	\$IRIS	Internal reader Initialization statement DSECT
\$IRWD	\$IRWD	Internal reader work area
\$IPCSWRK	\$IPCSWRK	IPCS work area DSECT
\$ITWORK	\$ITWORK	Initiator SSI Work Area DSECT
\$JAX	\$JAX	JOE Index access
\$JCMWORK	\$JCMWORK	JOB command PCE work area
\$JCT	\$JCT	JOB control table DSECT
\$JCTX	\$JCTX	JOB control table extension DSECT
\$JDSN	\$JDSN	JES2 job data set name DSECT
\$JAX	\$JAX	JOE Index access
\$JESLOG	\$JESLOG	JES Log Control Block
\$JFATAB	\$JFATAB	JOE field access table
\$JFL	\$JFL	JCL facility list DSECT
\$JFW	\$JFW	JCL facility work area DSECT
\$JIB	\$JIB	JOE information block DSECT
\$JNEW	\$JNEW	JESNEWS control block DSECT
\$JNT	\$JNT	Job number table DSECT
\$JOBENF	\$JOBENF	\$JOBENF Service Parameter List
\$JOE	\$JOE	Job output element DSECT
\$JOEIWRK	\$JOEIWRK	JOE Indexing PCE work area
\$JOT	\$JOT	Job output table DSECT
\$JPAWORK	\$JPAWORK	Job priority aging PCE work area
\$JQE	\$JQE	JOB queue element DSECT
\$JQRB	\$JQRB	JQE request block

*Table 8. JES2 DSECTIDs That Can Be Specified on \$MODULE (continued)*

<b>DSECTID</b>	<b>Macros</b>	<b>Description of Code Generated</b>
\$JQRWORK	\$JQRWORK	JQE request PCE work area
\$JRW	\$JRW	Job receiver work area
\$JSMT	\$JSMT	Job symbol table
\$JTW	\$JTW	Job transmitter work area
\$JVDTAB	\$JVDTAB	JOE View definition table
\$JVWA	\$JVWA	JOT verification work area DSECT
\$KAC	\$KAC	Checkpoint application copy control block
\$KAWA	\$KAWA	Checkpoint allocation work area
\$KEYLIST	\$KEYLIST	SWB keylist table entry DSECT
\$KIT	\$KIT	Checkpoint information table DSECT
\$LCK	\$LCK	Spool offload checkpoint element DSECT
\$LGRR	\$LGRR	LOGREC record SDWAVRA DSECT
\$LIMITS	\$LIMITS	Resource Limits control block DSECT
\$LMT	\$LMT	Load module table DSECT
\$LRC	\$LRC	Logical record DSECT
\$MCODE	\$MCODE	BSC code table DSECT
\$MCT	\$MCT	Master control table DSECT
\$MIGROBJ	\$MIGROBJ	Migration Object DSECT
\$MIT	\$MIT	Module information table DSECT
\$MITETBL	\$MITETBL	Module information table entry table DSECT
\$MLMWORK	\$MLMWORK	Line manager processor PCE work area DSECT
\$MODMAP	\$MODMAP	Module map directory DSECT
\$MONCB	\$MONCB	Monitor Address Space Control Block
\$MSCWORK	\$MSCWORK	Miscellaneous PCE Work Area DSECT
\$MSD	\$MSD	Monitor Sampling Data DSECT
\$MTQH	\$MTQH	Main task queue HEADER
\$MTRB	\$MTRB	Main task request block
\$MTL	\$MTL	I/O error recovery MTTR save table element
\$MWE	\$MWE	Monitor Work Element
\$NAT	\$NAT	Network nodes attached table
\$NCPE	\$NCPE	NJE common post element
\$NETACCT	\$NETACCT	Network ACCOUNT table format and DSECT
\$NHD	\$NHD	RK header DSECT
\$NHSB	\$NHSB	RK header spool block

Table 8. JES2 DSECTIDs That Can Be Specified on \$MODULE (continued)

DSECTID	Macros	Description of Code Generated
\$NIT	\$NIT	Network information table DSECT
\$NJETRC	\$NJETRC	NETSRV rolling trace area
\$NJEWORK	\$NJEWORK	NJE common work area
\$NJTWORK	\$NJTWORK	HASP network job transmitter work area
\$NMAP	\$NMAP	Network path manager notify map
\$NMR	\$NMR	Network communication message record DSECT
\$NOTENF	\$NOTENF	\$NOTENF Service Parameter List
\$NOUSWRK	\$NOUSWRK	Notify user message service area DSECT
\$NPMWORK	\$NPMWORK	Network path manager work area
\$NRD	\$NRD	\$NHDREAD parameter list
\$NRMWORK	\$NRMWORK	Network Resource Monitor Processor PCE Work Area
\$NSACT	\$NSACT	Network subnet anchor table entry DSECT
\$NSFP	\$NSFP	Network SWBTU functions parameter list DSECT
\$NSCT	\$NSCT	NETSRV address space control table
\$NSRWORK	\$NSRWORK	Network SYSOUT receiver PCE work area DSECT
\$NSST	\$NSST	NETSRV address space subtask table
\$NSTWORK	\$NSTWORK	Network SYSOUT transmitter PCE work area DSECT
\$NSWE	\$NSWE	NETSRV subtask work element
\$NSYWA	\$NSYWA	\$NITSYNC work area
\$NTK	\$NTK	Network path manager \$NATGET token
\$NTRDATA	\$NTRDATA	NJE Trace data area
\$NTW	\$NTW	Network path manager trace work area
\$NVL	\$NVL	Volume Allocation Table
\$NWR	\$NWR	\$NHDWRT parameter list
\$OFFSTBL	\$OFFSTBL	Offset table DSECT
\$OCR	\$OCR	Output control record DSECT
\$OCT	\$OCT	Output control table DSECT
\$ODPARM	\$ODPARM	Output descriptor parameter list DSECT
\$OPAWORK	\$OPAWORK	Output priority aging PCE work area
\$OUTWORK	\$OUTWORK	Output processor PCE work area DSECT
\$PAD	\$PAD	PROCLIB Allocation DSECT
\$PADDR	\$PADDR	Private area routine table
\$PAL	\$PAL	Page address list DSECT
\$PARMLST	\$PARMLST	Inline parameter list DSECT

*Table 8. JES2 DSECTIDs That Can Be Specified on \$MODULE (continued)*

<b>DSECTID</b>	<b>Macros</b>	<b>Description of Code Generated</b>
\$PARMWRK	\$PARMWRK	PARMLIB Work Area DSECT
\$PCE	\$PCE	Processor control element DSECT
\$PCETAB	\$PCETAB	PCE table (\$GETABLE) DSECT
\$PCIE	\$PCIE	Program controlled interrupt element DSECT
\$PCL	\$PCL	Persistent connection line element
\$PCT	\$PCT	Path manager control table DSECT
\$PCTAB	\$PCTAB	PC routine table (\$GETABLE) DSECT
\$PCYINTR	\$PCYINTR	Policy interpreter work area DSECT
\$PCYPARS	\$PCYPARS	Policy parser work area DSECT
\$PCYWORK	\$PCYWORK	Policy PCE work area DSECT
\$PDDB	\$PDDB	Peripheral data definition block DSECT
\$PERFCB	\$PERFCB	Performance data control block DSECT
\$PIT	\$PIT	Partitioned information table DSECT
\$PLX EQU	\$PLX EQU	PLX Equates
\$POLICY	\$POLICY	Policy control structures
\$PPPWORK	\$PPPWORK	Print/punch processor PCE work area DSECT
\$PQE	\$PQE	3800 page queue entry DSECT
\$PQH	\$PQH	3800 pending page queue header DSECT
\$PRA	\$PRA	Privileged resource management control block DSECT
\$PRE	\$PRE	Processor recovery element DSECT
\$PRGWORK	\$PRGWORK	Purge processor PCE work area DSECT
\$PRMD	\$PRMD	Process mode table entry DSECT
\$PSO	\$PSO	Process SYSOUT work area DSECT
\$PSOWORK	\$PSOWORK	PSO processor PCE work area DSECT
\$PSV	\$PSV	Process save area DSECT
\$QCT	\$QCT	Quickcell control table DSECT
\$QGET	\$QGET	QGET parameter list DSECT
\$QSE	\$QSE	Shared queue control element DSECT
\$QUEHEAD	\$QUEHEAD	Queue header DSECT
\$RAT	\$RAT	Remote attribute table DSECT
\$RCPWORK	\$RCPWORK	Remote console processor work area
\$RDRWORK	\$RDRWORK	Reader services PCE work area DSECT
\$RDT	\$RDT	Remote destination table DSECT
\$REQJID	\$REQJID	Request Job ID Specifications



Table 8. JES2 DSECTIDs That Can Be Specified on \$MODULE (continued)

DSECTID	Macros	Description of Code Generated
\$RESNAM	\$RESNAM	SAF resource name DSECT
\$RESWORK	\$RESWORK	Resource manager PCE work area DSECT
\$RECY	\$RECY	DAS Recovery CTENT DSECT
\$RGRPLST	\$RGRPLST	TREGROUP parameter list
\$RID	\$RID	Record identifier DSECT
\$RJCB	\$RJCB	Reader JOB card buffer DSECT
\$ROTT	\$ROTT	Rolling trace table DSECT
\$RVSTACK	\$RVSTACK	Error stack DSECT
\$RRTWA	\$RRTWA	Reroute authorization work area DSECT
\$RWL	\$RWL	Remote work look-up table
\$SAPID	\$SAPID	Sysout API data area
\$SBMT	\$SBMT	Submit Work Area DSECT
\$SBWA	\$SBWA	Spool browse work area
\$SAFINFO	\$SAFINFO	Security information parameter list
\$SCANTAB	\$SCANTAB	SCAN table (\$SCAN) DSECT
\$SCANWA	\$SCANWA	\$SCAN facility work area DSECT
\$SCAT	\$SCAT	SYSOUT class attribute table DSECT
\$SCID	\$SCID	Summary of checkpoint information DSECT
\$SCK	\$SCK	NJE TCP/IP Socket DSECT
\$SCQ	\$SCQ	Shared communication queue element DSECT
\$SCR	\$SCR	Spool control record DSECT
\$SCT	\$SCT	Spin communication table DSECT
\$SDB	\$SDB	Subsystem data set block DSECT
\$SFRB	\$SFRB	Scheduler facility request block DSECT
\$SFSWORK	\$SFRWORK	SJF services PCE work area DSECT
\$SFW	\$SFW	SWBTU functions work area DSECT
\$SIG	\$SIG	Spool signature record
\$SJB	\$SJB	Subsystem JOB block DSECT
\$SJXB	\$SJXB	Subsystem JOB extension block DSECT
\$SJIOB	\$SJIOB	Subsystem JOB input/output control block DSECT
\$SMF	\$SMF	SMF buffer DSECT
\$SNFWORK	\$SNFWORK	Spool management processor PCE work area DSECT
\$SPIWORK	\$SPIWORK	Sysout API PCE work area DSECT
\$SPMWORK	\$SPMWORK	Spool manager processor PCE work area DSECT

*Table 8. JES2 DSECTIDs That Can Be Specified on \$MODULE (continued)*

<b>DSECTID</b>	<b>Macros</b>	<b>Description of Code Generated</b>
\$SPNWORK	\$SPNWORK	Spin processor PCE work area DSECT
\$SPOOLCB	\$SPOOLCB	SPOOL Information Control Block
\$SQD	\$SQD	Subtask queue descriptor DSECT
\$SRW	\$SRW	SYSOUT receiver work area
\$STCWORK	\$STCWORK	Status/cancel PCE work area DSECT
\$STW	\$STW	SYSOUT transmitter work area
\$STWORK	\$STWORK	Subtask work area DSECT
\$SWBIT	\$SWBIT	SWB information table DSECT
\$SWBMPRM	\$SWBMPRM	SWBMERG parameter list DSECT
\$SWEL	\$SWEL	Sign on work element DSECT
\$SWR	\$SWR	SWB read parameter list
\$SXADDR	\$SXADDR	SCAN Exit Routine Address Table DSECT
\$SYMCB	\$SYMCB	Symptom record work area
\$S35D	\$S35D	WTO (SVC 35) work area DSECT
\$TAB	\$TAB	TRACK allocation block DSECT
\$TBUF	\$TBUF	TCP/IP Request Buffer DSECT
\$TED	\$TED	Trace enablement descriptor
\$TEWA	\$TEWA	Timed Event Work Area DSECT
\$TEXWORK	\$TEXWORK	Time excession monitor PCE work area
\$TGB	\$TGB	Allocation track group block DSECT
\$TIDTAB	\$TIDTAB	Trace ID table (\$GETABLE) DSECT
\$TIMWORK	\$TIMWORK	STIMER/TTIMER PCE Work Area
\$TINA	\$TINA	The indispensable non-volatile Array
\$TLBM	\$TLBM	Track Level Bit Map DSECT
\$TLGWORK	\$TLGWORK	Trace LOG processor PCE work area DSECT
\$TOR	\$TOR	Track one record DSECT
\$TOT	\$TOT	Track one table DSECT
\$TQE	\$TQE	Timer queue element format
\$TRCA	\$TRCA	Termination recovery control area DSECT
\$TRE	\$TRE	TCB recovery element DSECT
\$TRX	\$TRX	TCB recovery element extension DSECT
\$TTE	\$TTE	Trace table entry DSECT
\$TTETBL	\$TTETBL	TTE Trace Table DSECT
\$UPL	\$UPL	UCB parameter list DSECT

Table 8. JES2 DSECTIDs That Can Be Specified on \$MODULE (continued)

DSECTID	Macros	Description of Code Generated
\$URIMAP	\$URIMAP	URI mapping DSECT
\$USERCBS	\$USERCBS	User defined control blocks
\$VERTAB	\$VERTAB	Control block verification table DSECT
\$WARMCA	\$WARMCA	Warm start PCE communications area DSECT
\$WARMWRK	\$WARM	Warm start processor PCE work area DSECT
\$WAVE	\$WAVE	Work access verification element DSECT
\$WLMD	\$WLMD	Work Load Manager Data Bundle DSECT
\$WORK	\$WORK	\$GETWORK/\$RETNWORK general work area DSECT
\$WSA	\$WSA	Work selection area DSECT
\$WSC	\$WSC	WLM Service Class Queue Anchor
\$WSP	\$WSP	Work selection parameter area DSECT
\$WSTAB	\$WSTAB	Work selection table DSECT
\$XACTTAB	\$XACTTAB	Policy action implementation table DSECT
\$XBCWORK	\$XBCWORK	\$XBCAST parameter list DSECT
\$XCATAB	\$XCATAB	Work Selection Cache Attributes Table
\$XCBF	\$XCBF	CBFDSECT Work Area DSECT
\$XCMWORK	\$XCMWORK	XCF command processor PCE work area DSECT
\$XCW	\$XCW	SYSOUT Work Selection cache
\$XECB	\$XECB	Extended ECB element DSECT
\$XEQWORK	\$XEQWORK	Execution processor PCE work area DSECT
\$XFATAB	\$XFATAB	Policy attribute implementation table DSECT
\$XFMWORK	\$XFMWORK	XFR I/O manager processor PCE work area DSECT
\$XIT	\$XIT	Exit information table DSECT
\$XMAS	\$XMAS	XCF cross MAS coupling block
\$XOPTAB	\$XOPTAB	Policy operator parsing table DSECT
\$XPL	\$XPL	Exit parameter list DSECT
\$XPWORK	\$XPWORK	XCF processor work area DSECT
\$XREQ	\$XREQ	XCF information request message
\$XRQ	\$XRQ	XCF group exit request block
\$XRT	\$XRT	EXIT routine table DSECT
\$XTREE	\$XTREE	Binary tree services
\$XVCTAB	\$XVCTAB	Policy name resolution parsing table DSECT
\$YLG	\$YLG	Instream substitution log control
\$ZGL	\$ZGL	Job group logging

Table 8. JES2 DSECTIDs That Can Be Specified on \$MODULE (continued)		
DSECTID	Macros	Description of Code Generated
\$ZJC	\$ZJC	Zone Job Container DSECT
<ul style="list-style-type: none"><li>• \$USERCBS, as received from IBM, is null. Installations can use it to provide their own control block mappings.</li><li>• When you use \$MODULE to prepare a non-IBM JES2 module (for example, an installation-written exit routine), \$MODULE always expands the mapping of \$USERCBS.</li></ul>		

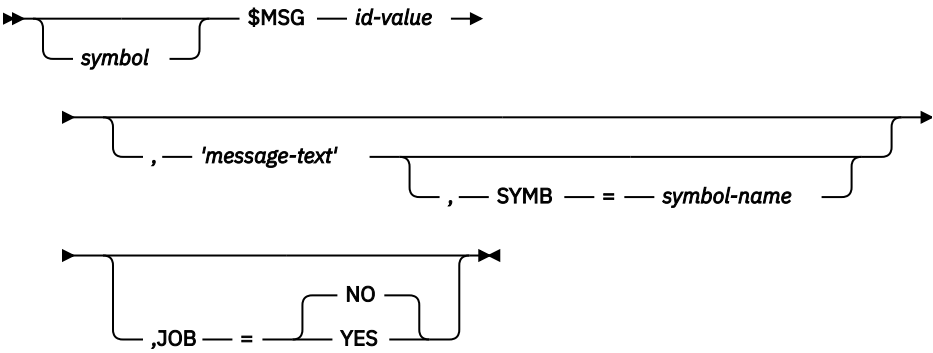
Environment

- JES2 main task, JES2 subtask, user, or FSS.
- MVS WAIT and \$WAIT are not applicable.

\$MSG – Write to operator message area

Use \$MSG to generate a message text area to be referenced by the message operand of the \$WTO macro instruction.

Format description



- id**  
Specifies the numeric 3-digit message identification that is to be displayed with the message text.
- message**  
Specifies the character string enclosed within single quotation marks that is to be displayed as the informational portion of the message. If the purpose of this macro instruction is to generate only the message identification, this operand should be omitted.
- SYMB=**  
Specifies the symbol-name that is to be assigned to the message text portion of the area generated by the macro instruction. If this operand is specified, the message operand must be specified. This symbol may be used to modify variable portions of the message text before executing the corresponding \$WTO macro instruction. It must not be referred to directly by the \$WTO macro instruction; the symbol assigned to the beginning of the area must be used for this purpose.
- JOB=**  
Specifies whether the user, at \$WTO macro execution time, has placed the 18-byte job identification information into the beginning of the text portion using the symbol as specified by the SYMB operand. The format of the job identification is as follows:
- Byte**  
**Content**  
**0-7**  
Job identification (JOBnnnn, STCnnnn, or TSUnnnn)

8

Blank

9-16

Job name

17

Blank

Specifications for the JOB operand are as follows:

**YES**

The user places job information into the message text portion of the area before executing a \$WTO macro instruction.

**NO (default)**

The user does not place job information into the message area but can require the \$WTO macro instruction to extract job information from the job control table and append the information to the console message buffer copy of the message during \$WTO macro execution time.

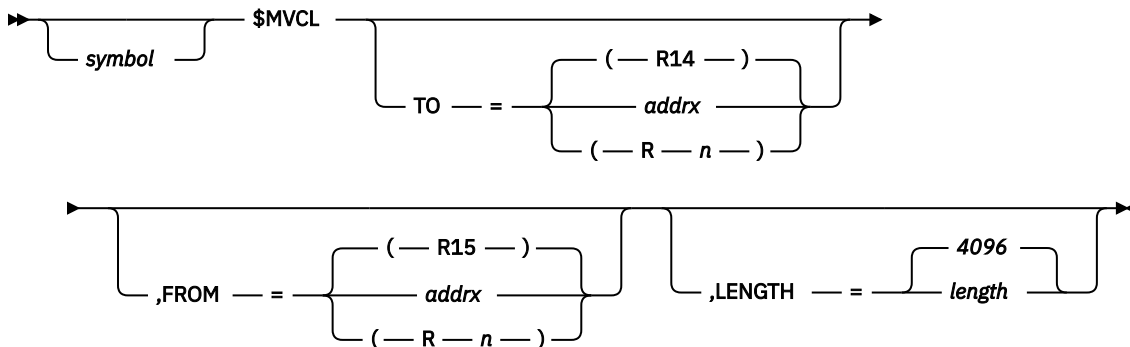
## Environment

- Main task.
- \$WAIT cannot occur.

## \$MVCL – Move more than 256 bytes of storage

Use \$MVCL to generate a MVC (move character) instruction when you need to move more than 256 bytes of storage. Use this macro instruction in high performance areas because multiple MVCs (as created by this macro) are faster than using an MVCL instruction.

## Format description

**TO=**

Specifies an address or register containing the address of the area to which the storage area is to be moved. Any register, except R0, can be specified. If TO= specifies a value, that value is loaded into R14.

**Note:** If you do not code this keyword, the value currently in R14 is used.

**FROM=**

Specifies an address or register containing the address of the storage area to be moved. Any register, except R0, can be specified. If FROM= specifies a value, that value is loaded into R15.

**Note:** If you do not code this keyword, the value currently in R15 is used.

**LENGTH=**

Specifies the length (in bytes) of the storage area to be moved. If you do not specify a value, 4K (4096) bytes will be moved. Any length up to a maximum of 4096 bytes can be specified. The value specified here must be a hard-coded value.

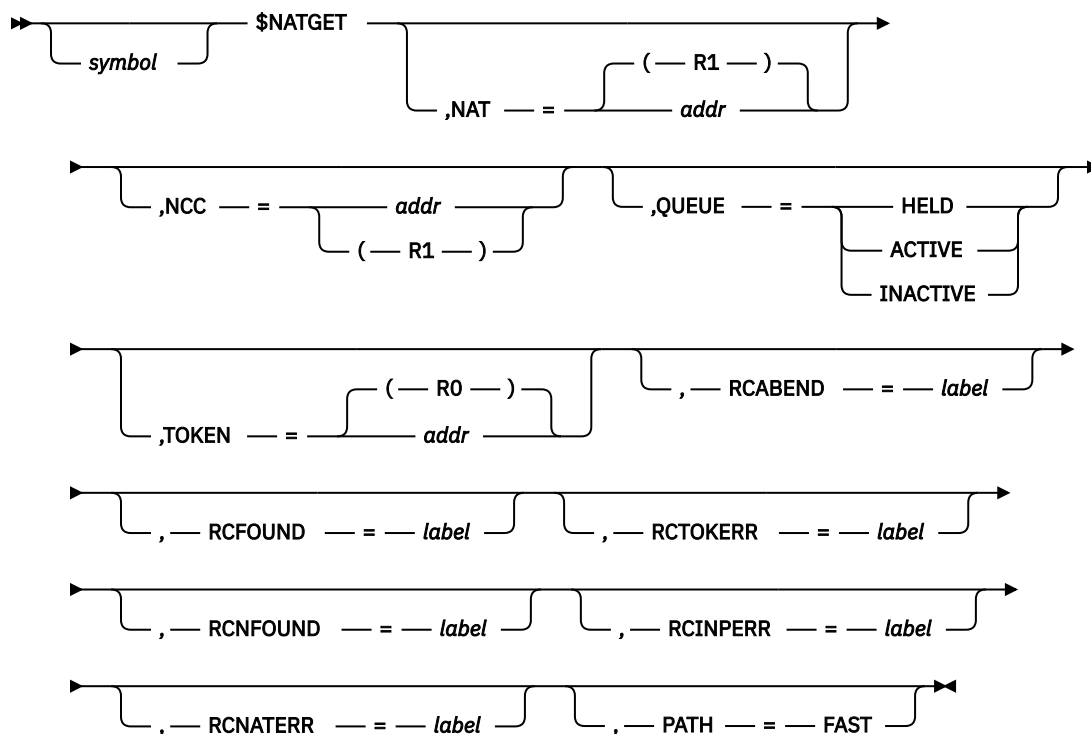
## Environment

- Main task, subtask, user, and functional subsystem (HASPFSM).
- \$WAIT cannot occur.

## \$NATGET – Locate a NAT element

Use \$NATGET to locate elements in the nodes attached table (NAT).

### Format description

**NAT=**

Specifies the address of a prototype NAT element that is to be located in the NAT. The address of the real NAT element will be returned in register 1. This parameter may not be specified with the NCC= parameter.

**NCC=**

Specifies the address of a prototype NCC record that is to be located in the NAT. The address of the real NAT element will be returned in register 1. This parameter may not be specified with the NAT= parameter.

**QUEUE=**

Specifies the queues upon which NAT elements may be found.

**HELD**

Specifies that the HELD queue is to be searched for the NAT element.

**ACTIVE**

Specifies that the ACTIVE (CONNECTED) queue is to be searched for the NAT element.

**INACTIVE**

Specifies that the INACTIVE (UNCONNECTED) queue is to be searched for the NAT element.

**TOKEN=**

Specifies the address of a token to be used when chaining through the NAT elements for all connections for a particular node. On the first call to \$NATGET, the NAT=, and NCC=, or QUEUE=

parameter should be specified with the TOKEN= parameter, to get the address of the first NAT on the chain. On subsequent calls, only the TOKEN= parameter needs to be specified.

**RCABEND=**

Specifies a label to which control should be passed if a NAT element could not be found because of an ABEND in the \$NATGET service.

**RCFOUND=**

Specifies a label to which control should be passed if the NAT element was successfully located in the NAT. The default is to pass control to the next sequential instruction after the \$NATGET.

**RCTOKERR=**

Specifies a label to which control should be passed if an error was detected in the token passed on the TOKEN= parameter. The default is to pass control to the next sequential instruction after \$NATGET.

**RCNFOUND=**

Specifies a label to which control should be passed if a NAT element matching the prototype could not be found, or if no more NAT elements matching the prototype could be found if TOKEN= was specified. The default is to pass control to the next sequential instruction after \$NATGET.

**RCINPERR=**

Specifies a label to which control should be passed if a NAT element could not be found because the input passed to the \$NATGET routine was not valid. A reason code will be passed back in register 0 indicating what error was detected. The default is to pass control to the next sequential instruction after \$NATGET.

**RCNATERR=**

Specifies a label to which control should be passed if a NAT element could not be found because there was an error in the NAT. The default is to pass control to the next sequential instruction after \$NATGET.

**PATH=**

Specifies that a “fast path” is to be taken through the \$NATGET service routine. This fast path call should be used only by main line path manager code, as it bypasses certain error checking and does not set an \$ESTAE. If \$NATGET is issued from an environment other than JES2 main task, PATH=FAST must be specified.

## Return codes

**0**

Indicates that the NAT element was found.

**4**

Indicates that the value on the TOKEN= parameter was not valid.

**8**

Indicates that the NAT element was not found.

**12**

Indicates that an input error was detected. One of the following reason codes will be returned in register 0:

**0**

NAT, NCC, or TOKEN address was required but was not specified.

**4**

Error in primary node specification.

**8**

Error in primary member specification.

**12**

Error in secondary node specification.

**16**

Error in secondary member specification.

**28**

Status error.

**16**

Indicates that an error was detected in the nodes attached table.

**20**

Indicates that the \$NATGET service abended and recovered.

## Environment

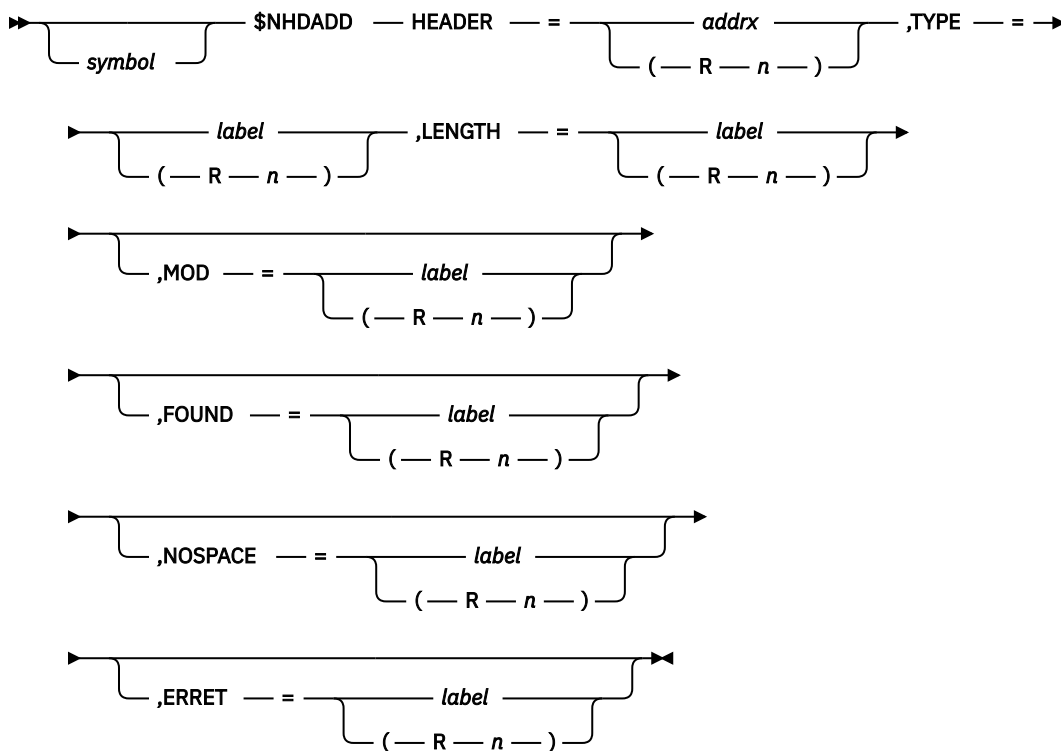
- JES2 main task or subtask.
- \$WAIT cannot occur.

## \$NHDADD – Adds an installation-defined section to an NJE data area

Use the \$NHDADD macro to add an installation-defined section to an NJE data area. An NJE data area can be one of the following:

- NJE job header
- NJE data set header
- NJE job trailer

## Format description



### HEADER=

Specifies the address of the NJE job header, NJE data set header, or NJE job trailer to which the installation-defined section should be added. The installation-defined section to be added to the NJE data area is defined by the TYPE= and MOD= parameters. This is a required parameter.

### TYPE=

Specifies the type NJE data area to which you want to add an installation-defined section.



- NJHUTYPE if the installation-defined section is to be added to the job header.
- NDHUTYPE if the installation-defined section is to be added to the data set header.
- NJTUTYPE if the installation-defined section is to be added to the job trailer.

This is a required parameter.

#### **FOUND=**

Specifies the label where your routine should continue processing when the installation-defined section you were attempting to add was already contained in the NJE data area. If you do not provide an error routine when the installation-defined section was already contained in the NJE data area, JES2 will continue to process the NJE job.

This is an optional parameter. If you do not specify this parameter and the installation defined section already exists in the NJE data area, JES2 will continue processing the NJE data area.

#### **MOD=**

Specifies the modifier of the installation-defined section you want to add to the NJE data area.

- NJHUMOD if you are adding an installation-defined section to the NJE job header.
- NDHUMOD if you are adding an installation-defined section to the NJE data set header.
- NJTUMOD if you are adding an installation-defined section to the NJE job trailer.

This is an optional parameter.

#### **LENGTH=**

Specifies the length of the installation-defined section to be added to the NJE data area.

This is a required parameter.

#### **NOSPACE=**

Specifies the label where your routine should continue processing when the installation-defined section you were attempting to add caused the NJE data area to exceed the maximum length. If you do not provide an error routine when JES2 could not add the installation-defined section to the NJE data area, JES2 will continue to process the NJE job.

This is an optional parameter. If you do not specify this parameter and the installation-defined section caused the NJE data area to exceed the maximum length, JES2 will not add the installation-defined section and will continue to process the NJE data area.

#### **ERRET=**

Specifies the label or register that contains the address of an error routine that receives control if JES2 could not locate the NJE data area specified by the HEADER parameter. If you do not provide an error routine when JES2 cannot locate the header, JES2 will continue to process the NJE job. JES2 returns one of the following return codes in register 15. IBM suggests that you use the NOSPACE, FOUND, and ERRET parameters if you are going to code routines for the conditions indicated by the return codes.

## **Return codes**

The following return codes (in decimal) are returned in register 15.

#### **Return Code**

#### **Meaning**

**0**

Indicates the installation-defined section was added to the NJE data area.

**4**

Indicates the installation-defined section was not added to the NJE data area because it already existed. Ensure you have specified the correct value for the TYPE= and MOD= parameters.

**8**

Indicates the installation-defined section was not added to the NJE data area because it would make the header exceed the length of the NJE data area.

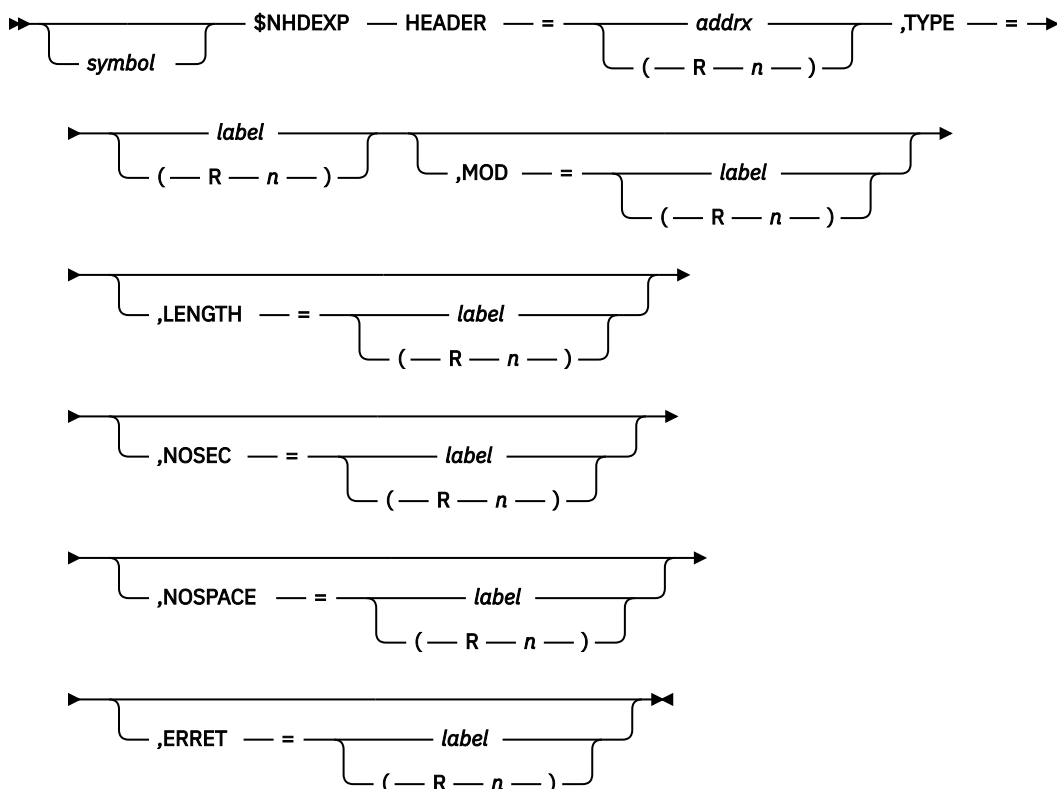
**12**

Indicates the installation-defined section was not added to the NJE data area because the address specified on the HEADER= parameter was not valid.

## **\$NHDEXP – Expand an NJE data area**

Use the \$NHDEXP macro to expand an installation-defined section to either the NJE job header or NJE data set header.

### **Format description**

**HEADER=**

Specifies the address of the NJE job header, NJE data set header, or NJE job trailer that needs to be expanded to contain additional information. Your installation should be expanding only the installation-defined section of the NJE data area. This is a required parameter.

**TYPE=**

Specifies the type of installation-defined section you want to expand. You can obtain the value to specify for this parameter from one of the following:

- NJHUTYPE if the installation-defined section of the NJE job header needs to be expanded.
- NDHUTYPE if the installation-defined section of the NJE data set header needs to be expanded.
- NJTUTYPE if the installation-defined section of the NJE job trailer needs to be expanded.

This is a required parameter.

**MOD=**

Specifies the modifier of the installation-defined section you want to expand. This is an optional parameter.

**LENGTH=**

Specifies the length of the area you want to add to the NJE data area.

This is a required parameter.

**NOSEC=**

Specifies the label or register that should receive control if the installation-specific section could not be expanded because it did not exist in the NJE data area.

**NOSPACE=**

Specifies the label or register that should receive control if the installation-specific section that you attempted to expand exceeded the maximum size of the NJE data area.

This is an optional parameter if the ERRET= keyword is not specified.

**ERRET=**

Specifies the label or register that contains the address of an error routine that receives control if an error occurs while expanding an NJE data area.

## Return codes

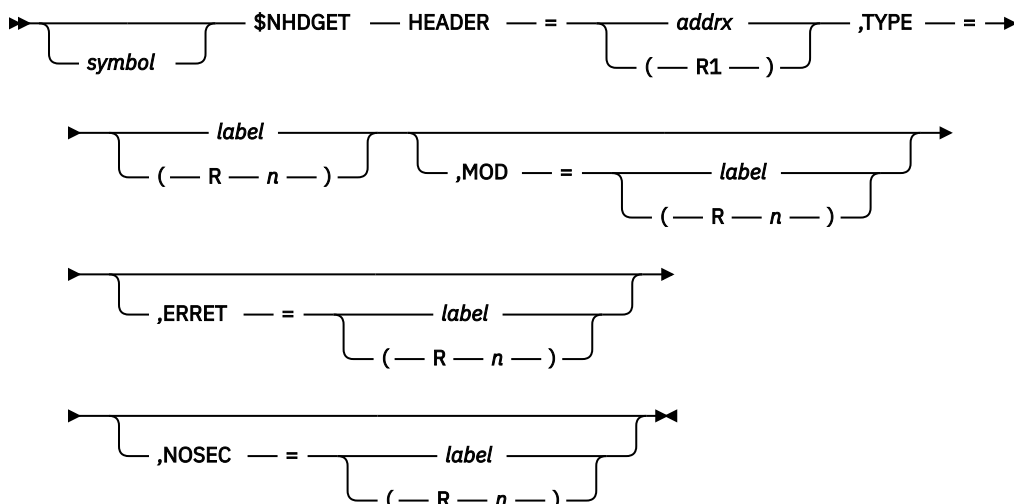
The following return codes (in decimal) are returned in register 15.

Return Code	Description
0	Indicates the networking job header or networking data set header was expanded to include the installation-defined section.
4	Indicates the NJE data area was not expanded because JES2 could not locate the installation-defined section.
8	Indicates the NJE data area was not expanded because if the installation-defined section was added to the NJE data area it would cause it to exceed the maximum length.
12	Indicates the NJE data area was not expanded because an invalid NJE data area was specified.

## \$NHDGET – Get the network header section

Use \$NHDGET to search the network job header, job trailer, or data set header to locate a specified section of a control block. If the control block section is located, the address is returned in register 1. This address can then be used to access the control block section if you need to modify that header or trailer information.

### Format description



**HEADER=**

Specifies the address of the control block within the section indicated by the TYPE= keyword is to be located.

**TYPE=**

Specifies an 8-bit mask which indicates the type of section to be located. Valid types and their corresponding masks are defined in the \$NHD macro (the job header DSECT). TYPE= can be specified as either a register whose low-order byte contains the type mask, or TYPE= can be specified as a label that is equated to a 1-byte mask.

**MOD=**

Specifies an 8-bit mask which indicates the value of the modifier field of the control block section to be located. MOD= can be specified as either a register whose low-order byte contains the type mask, or MOD= can be specified as a label is equated to a 1-byte mask. The valid bit mask settings are also located in the \$NHD macro.

**ERRET=**

Specifies a label or register of an error routine which receives control if the specified header type is not valid.

**NOSEC=**

Specifies a label or register of an error routine which receives control if the specified header type is not found.

**Note:**

1. Register 1 contains the address of the control block section if it is located.
2. Registers 0, 1, and 15 are used by this macro; do not use them.

## Environment

- All environments.
- \$WAIT cannot occur.

## \$NHDREM – Removes an installation-defined section from an NJE data area

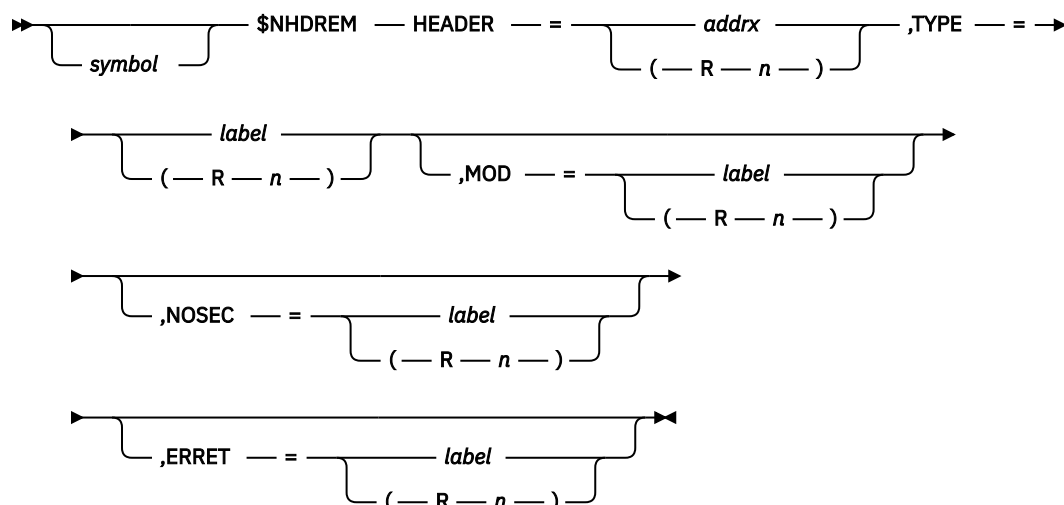
---

Use the \$NHDREM to remove an installation-defined section from an NJE data area. An NJE data area can be one of the following:

- NJE job header
- NJE data set header
- NJE job trailer

If your installation uses the \$NHDADD macro to add any installation-defined sections to any of the NJE data areas, you might need to issue a \$NHDREM to remove the installation-defined sections before transmitting the NJE job.

## Format description



### HEADER=

Specifies the storage address of the NJE job header, NJE data set header, or NJE job trailer from which the installation-defined section should be removed. The section is identified by the TYPE= and MOD= parameters.

### TYPE=

Specifies the type of NJE data area from which you want to remove the installation-specific section. You can obtain the value for this parameter from one of the following fields:

- **NJHUTYPE** if the installation-defined section is to be removed from the job header.
- **NDHUTYPE** if the installation-defined section is to be removed from the data set header.
- **NJTUTYPE** if the installation-defined section is to be removed from the job trailer.

### MOD=

Specifies the modifier associated with the installation-defined section to be removed from the NJE data area. You can obtain the value for this parameter from one of the following fields:

- **NJHUMOD** if the installation-defined section is to be removed from the job header.
- **NDHUMOD** if the installation-defined section is to be removed from the data set header.
- **NJTUMOD** if the installation-defined section is to be removed from the job trailer.

### NOSEC=

Specifies the label or register that should receive control if the section specified to be deleted is required by NJE protocols or JES2 could not locate the specified section.

### ERRET=

Specifies the label or register that contains the address of an error routine that receives control if an error occurs because JES2 could not locate the NJE data area specified by the HEADER parameter.

## Return codes

The following return codes (in decimal) are returned in register 15:

Return Code	Meaning
-------------	---------

0	JES2 removed the requested section from the NJE data area
---	---

**4**

JES2 did not remove the requested section from the NJE data area because it could not locate the specified section or because NJE protocols prohibit the section from being removed from the NJE data area.

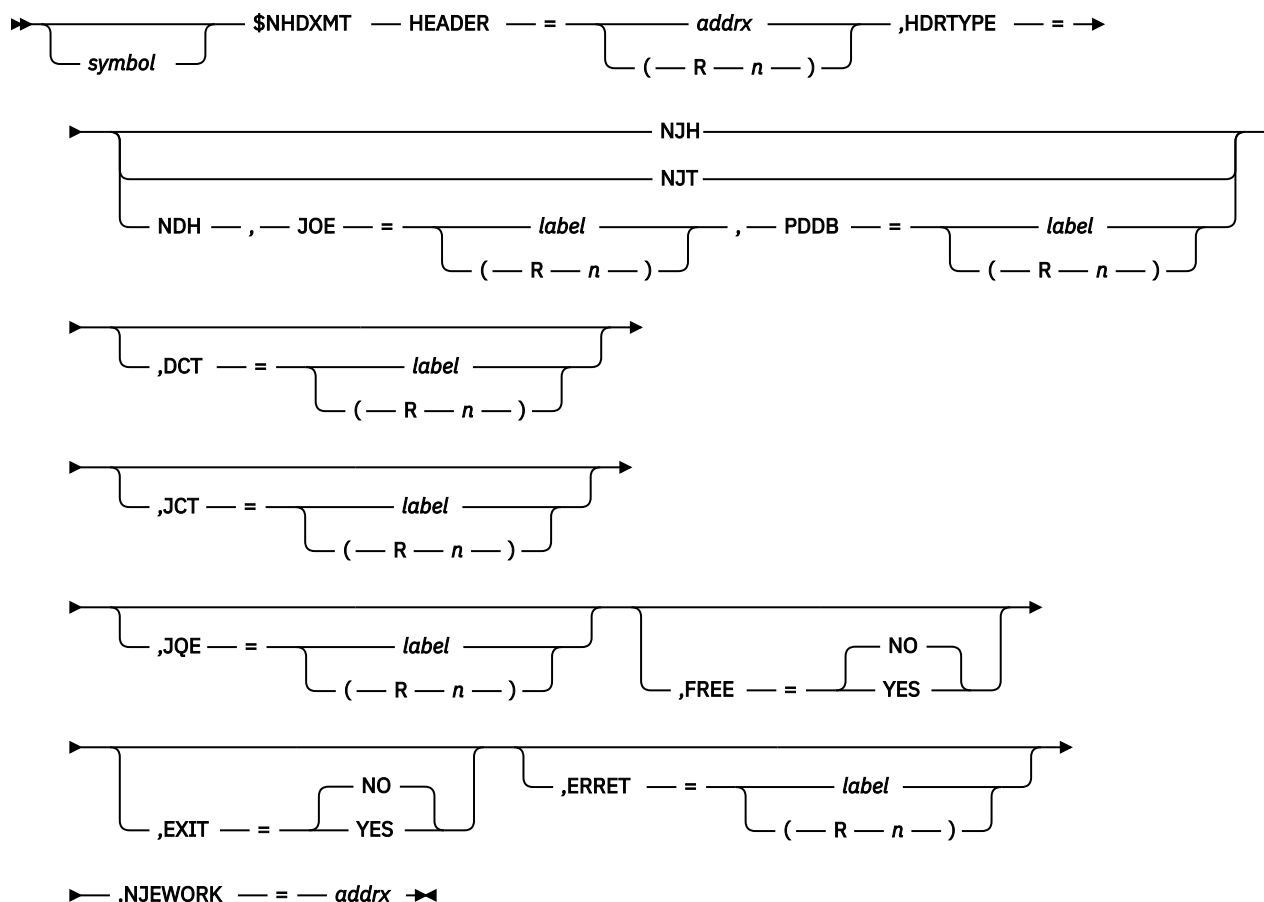
**8**

JES2 did not remove the requested section from the NJE data area because it could not locate the NJE data area.

## **\$NHDXMT – Transmitting an NJE data area across the network**

Use \$NHDXMT to transmit an NJE data area to another node in the network.

### **Format description**



#### **HEADER=**

Specifies the address of the buffer containing the NJE data area to be transmitted. The HEADER= keyword is required.

#### **DCT=**

Specifies the address of the \$DCT. The DCT= keyword is required.

#### **JQE=**

Specifies the address of the \$JQE. The JQE= keyword is a required keyword.

#### **JCT=**

Specifies the address of the \$JCT. The JCT= keyword is required.

#### **PDDB=**

Specifies the address of the \$PDDB that is associated with the SYSOUT data set being transmitted. The PDDB= keyword is required when transmitting data set header.

**JOE=**

Specifies the address of the \$JOE that is associated with the SYSOUT data set being transmitted. The JOE= keyword is required when transmitting data set header.

**FREE=**

Specifies whether (YES) or not (NO) the buffer that contains the NJE data area should be freed after it is transmitted. The default for the FREE parameter is NO.

**EXIT=**

Specifies whether (YES) or not (NO) exit 46 should be invoked. The default is NO. You should not specify YES on the EXIT= keyword if you are issuing the \$NHDXMT macro in exit 46.

**HDRTYPE=**

Specifies the type of NJE data area to be transmitted. You can specify one of the following values:

**Value****Meaning****NJH**

Networking job header

**NJT**

Networking job trailer

**NDH**

Networking data set header

**ERRET=**

Specifies the label or register that contains the address of an error routine that receives control if an error occurs while reading an NJE data area from spool. JES2 indicates whether or not the NJE data area was successfully transmitted by returning a return code in register 15.

**NJEWORk**

Specifies the address of the JTW or STW. This keyword is required.

## Return codes

The following return codes (in decimal) are returned in register 15:

**Return Code****Meaning**

**0**

JES2 successfully transmitted the NJE data area to another node.

**4**

An error occurred while transmitting the NJE data area to another node.

## Environment

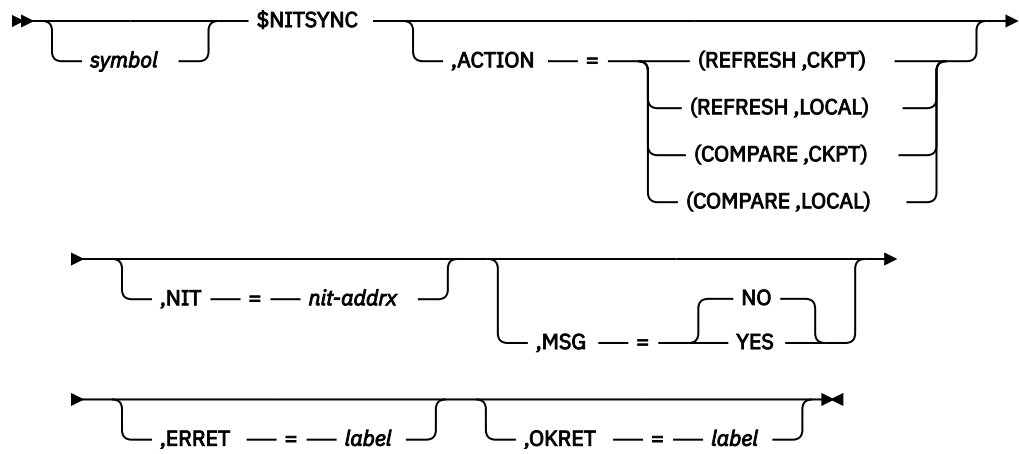
- JES2 Main task.

## \$NITSYNC – Synchronize NIT settings

---

Use \$NITSYNC to synchronizes the private checkpointed NITs and all associated information.

Format description



**ACTION=**

Specifies the action that is to be taken:

**(REFRESH,CKPT)**

Indicates that the checkpointed NITCs are to be updated from the data in the private NITs.

**(REFRESH,LOCAL)**

Indicates that the local node definitions are to be updated from the checkpoint

**(COMPARE,CKPT)**

Indicates that the local and checkpointed definitions are to be compared, and messages issued if appropriate, but no action is to be taken. The comparison assumes that the CKPT fields are potentially downlevel.

**(COMPARE,LOCAL)**

Indicates that the local and checkpointed definitions are to be compared, and messages issued if appropriate, but no action is to be taken. The comparison assumes the LOCAL fields are potentially downlevel.

**NIT**

Specifies that an optional NIT address to be updated. If specified, the definitions for this node will be updated. If not specified, all nodes are to be processed.

**MSG=**

Indicates whether (YES) or not (NO) a message should be issued for each node definition that needs to be updated. The default is NO.

**ERRET**

Label to receive control when changes are needed.

**OKRET**

Label to receive control when changes are not needed.

Return codes

The following return codes (in decimal) are returned in register 15:

Return Code	Meaning
-------------	---------

- |   |  |
|---|--|
| 0 | Processing successful, no changes were needed.         |
| 4 | Processing successful, at least one change was needed. |



## Environment

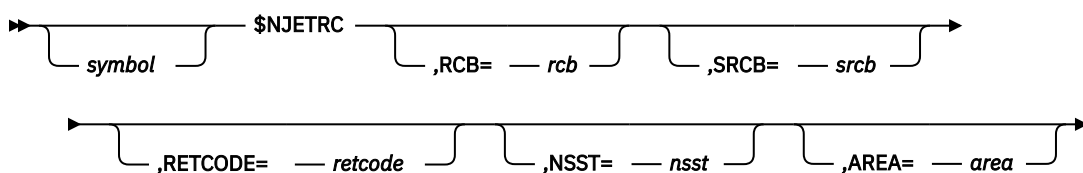
- JES2 Main task.

## \$NJETRC - NJE subdevice rolling trace

---

\$NJETRC maps the rolling trace area associated with a particular subdevice. NJETRC is also used to invoke the trace routine itself by specifying TYPE=TRACE.

### Format description



#### RCB=

Points to the RCB of the data to be traced.

#### SRCB=

Points to the SRCB of the data to be traced.

#### RETCODE =

The return code being passed back to IAZNJTCP.

#### NSST=

Points to the NSST associated with the line. You can use this to generate the correct sequence number for the trace.

#### AREA=

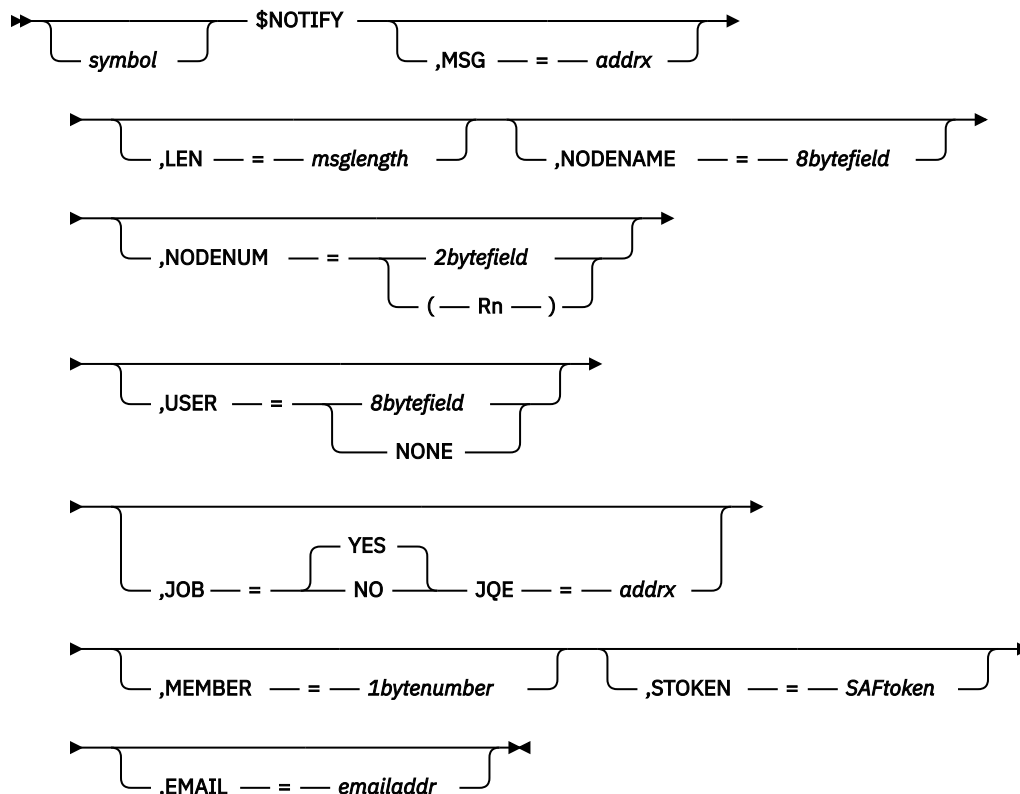
Points to the NJETRC area into which the trace should be merged.

## \$NOTIFY – Send a notify message to a specific user ID and node

---

Use \$NOTIFY to send a notify message to a specific user ID and node.

## Format description



### MSG=

Specifies the address of the message to be sent in the form that is generated by the \$MSG macro, in the following format: XL2'nnnF',C'message text'.

### LEN=

Specifies the length of the message. This parameter defaults to the length of the field specified by MSG=.

### NODENAME =

Specifies an 8-byte field containing the node name to which the message is to be sent.

### NODENUM=

Specifies a 2-byte field containing the node number to which the message is to be sent, or a register containing the node number.

### USER=

Specifies an 8-byte field containing the userid to which the message is to be sent. If USER=NONE is specified, the message is queued to the master console on the specified node.

### JOB=

Specifies whether (YES) or not (NO) job name and job id information are to be added to the message from either PCEJQE (main task) or JSAB (other environments). The default is JOB=YES.

### JQE=

Specifies the address of a JQE with the job name and job id to be placed in the message. This parameter is optional and is only allowed when JOB=YES.

### MEMBER=

Specifies a 1-byte member number that indicates the member that the CMB containing the notify message should be routed to if the user is not logged on and shared broadcast is not used (OUTDEF BRODCAST=NO).

### EMAIL=

Alternative way to specify notification target. If specified, requests email notification rather than a TSO message. EMAIL specifies an address of a chain of email addresses in the format that is expected

by the Job Notify SSI (SSI 75). See *z/OS MVS Data Areas*, in the *z/OS Internet library* ([www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary](http://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary)), for a description of SSNUADDR field in the IAZSSNU macro.

When specified, NODENAME, NODENUM, USER, and MEMBER are not allowed.

**STOKEN=**

Specifies the address of a security facility token to validate authority to send a message. If not specified, the authority of the caller is used. This parameter is optional.

**Return codes**

The following return codes (in decimal) are returned in register:

**Return Code**  
**Meaning**
**0**

The message was issued.

**4**

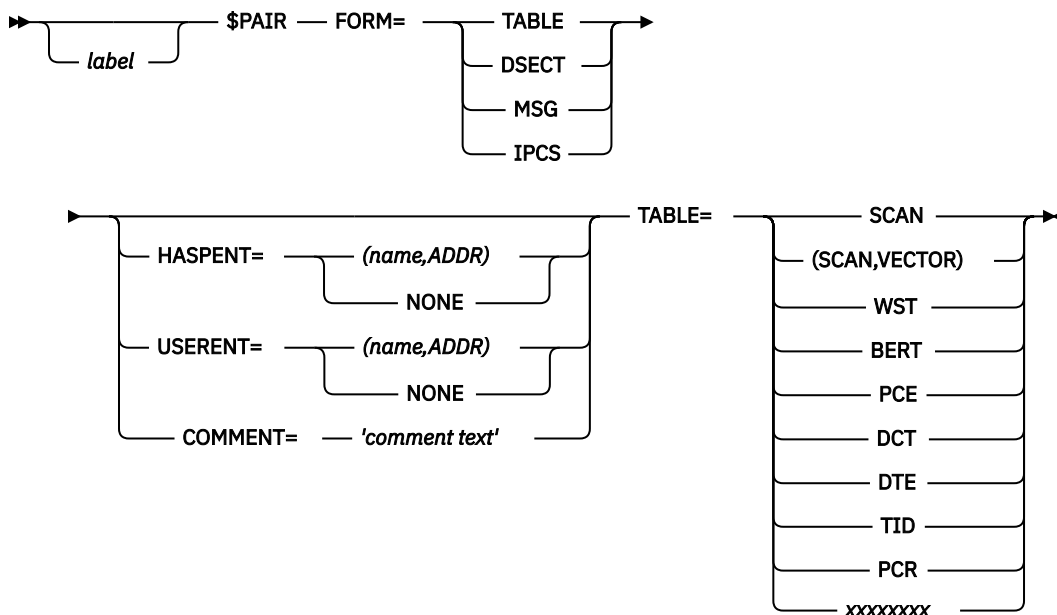
The message was not issued.

**Environment**

Any environment (JES2, USER, SUBTASK, FSS)

**\$PAIR – Define a table pair**

Use \$PAIR to define a table pair. See [Appendix A, “Using JES2 table pairs,” on page 425](#) for a detailed description of table pairs.

**Format description****FORM=**

Indicates what of several forms of the table pair is being generated:

**DSECT**

Generates the DCs required for the DSECT mapping.

## TABLE

**IPCS**

**MSG**

**HASPENT=**

**USERENT=**

COMMENT=

TABLE=

TABLE=SCAN - \$SCANTAB  
TABLE=(SCAN,VECTOR) - \$SCANTAB VECTOR  
TABLE=WST - \$WSTAB  
TABLE=BERT - \$BERTTAB  
TABLE=PCE - \$PCETAB  
TABLE=DTE - \$DTETAB  
TABLE=DCT - \$DCTTAB  
TABLE=TID - \$TIDTAB  
TABLE=PCR - \$PCTAB

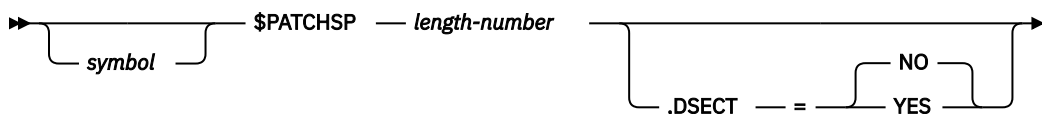
## Environment

- Main task.
- \$WAIT can occur.

## \$PATCHSP – Generate patch space

Use \$PATCHSP to cause a specified number of bytes of patch space to be generated. This patch space is divided into halfwords and listed in the assembly in such a way that both the assembly location (for REP and AMASPPAP patch statements) and the base displacement (in the form BDDD) are printed for each halfword.

## Format description



## length

Specifies the length of the patch space in bytes.



**\$PCEDYN**

**CENTER=**

Specifies how the separator page block letters are placed on the page as follows:

**YES (default)**

The block letters are to be centered. However, if the field containing the character string is filled with trailing zeros rather than blanks, the block letters will not be centered.

**NO**

The block letters are to be left-justified.

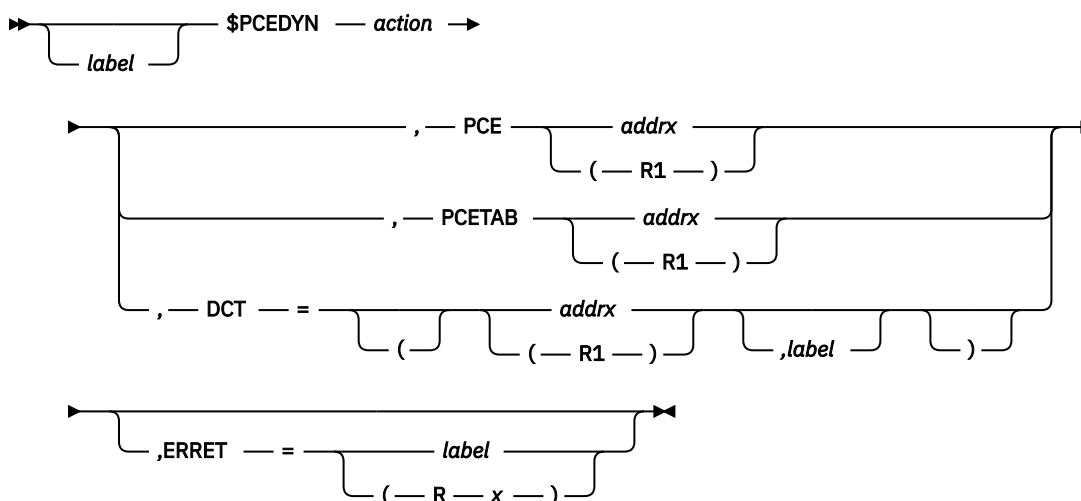
## Environment

- Main task.
- \$WAIT can occur.

## \$PCEDYN – Attach or delete a JES2 PCE

This macro provides the interface to those services that perform all generating (ATTACHing) and deletion (DETACHing) of JES2 processors (PCEs).

## Format description



**action**

Specifies the type of action requested.

## ATTACH

Generate a new PCE(s). These PCEs are dispatched based on the DISPTCH keyword on the associated \$PCETAB entries.

## DETACH

Dequeue and delete a PCE(s).

## DETACHTEST

Determine if a PCE(s) can be detached now.

**PCE=**

Specifies the address of a PCE. This address can be either a register (1-12) or the name of a field containing the PCE address. R1 is loaded with the value.

**If Action is:**

**This keyword indicates:**

**ATTACH**

Specifies an existing PCE of a PCE type (PCEID value) for which another PCE (1) will be generated.

**DETACH**

Specifies a PCE to detach.

**DETACHTEST**

Specifies to test a PCE and determine if this PCE can be deleted (that is, determine if any resources are still outstanding).

**PCETAB=**

Specifies the address of a PCE table entry for a PCE type. This address can be either a register (1-12) or the name of a field containing the entry address.

**If Action is:**

**This keyword indicates:**

**ATTACH**

Specifies a table entry for a PCE type that is not one-to-one with a DCT type. A PCE of this type is to be generated.

**DETACH**

PCETAB= cannot be specified for DETACH. \$PCEDYN cannot detach an arbitrary PCE of a PCE type.

**DETACHTEST**

PCETAB= cannot be specified for DETACH. \$PCEDYN cannot detach an arbitrary PCE of a PCE type.

**DCT=**

Specifies the address of a device control table (DCT). This address can be either a register (1-12) or the name of a field containing the DCT address. R1 is loaded with the value.

Also, to indicate a DCT chain field that should be used to attach or detach PCEs for the DCT chain starting with the DCT specified, use a second positional parameter. When a DCT chain is to be processed, \$PCEDYN will attach or detach PCEs, connect or disconnect DCTs and their 'managing' PCEs, or do nothing as indicated by the associated DCT and PCE table entries (for example, a DCT not owned or managed by a specific processor).

**Note:**

1. If a DCT chain is specified, no PCE is attached for DCTs for which DCTPCE is already nonzero.
2. If any PCE ATTACH fails for a DCT in the chain, the entire DCT chain is processed as if DETACH had been requested.

**ERRET=**

Defines an error routine location (label or register) to branch to if R15 is not zero on return from ATTACH.

**Attention:**

Before using this macro, it may be useful to review the table structures involved, that is, the PCE table and the DCT table in HASPTABS (macros \$PCETAB and \$DCTTAB). Also, the mapping macros for the PCE (\$PCE) and DCT (\$DCT) provide further understanding of these control blocks.

## Environment

- JES2 Main task.
- \$WAITs cannot occur.

## \$PCETAB – Generate or map PCE table entries

---

Use \$PCETAB to map and generate PCE table entries.

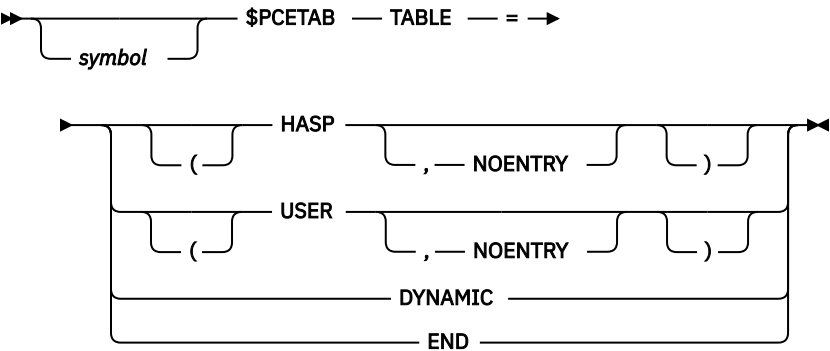
\$PCETAB entries are used to define the start of a user table (\$PCETAB TABLE=USER...) or a JES2 table (\$PCETAB TABLE=HASP...), the end of a table (\$PCETAB TABLE=END) or an entry in a table (\$PCETAB NAME=ZOOT...).

**Note:** The format description that follows breaks the macro into a **boundary form** (the form that starts or ends a table) and an **entry form** (the form that defines each table entry).

Format description

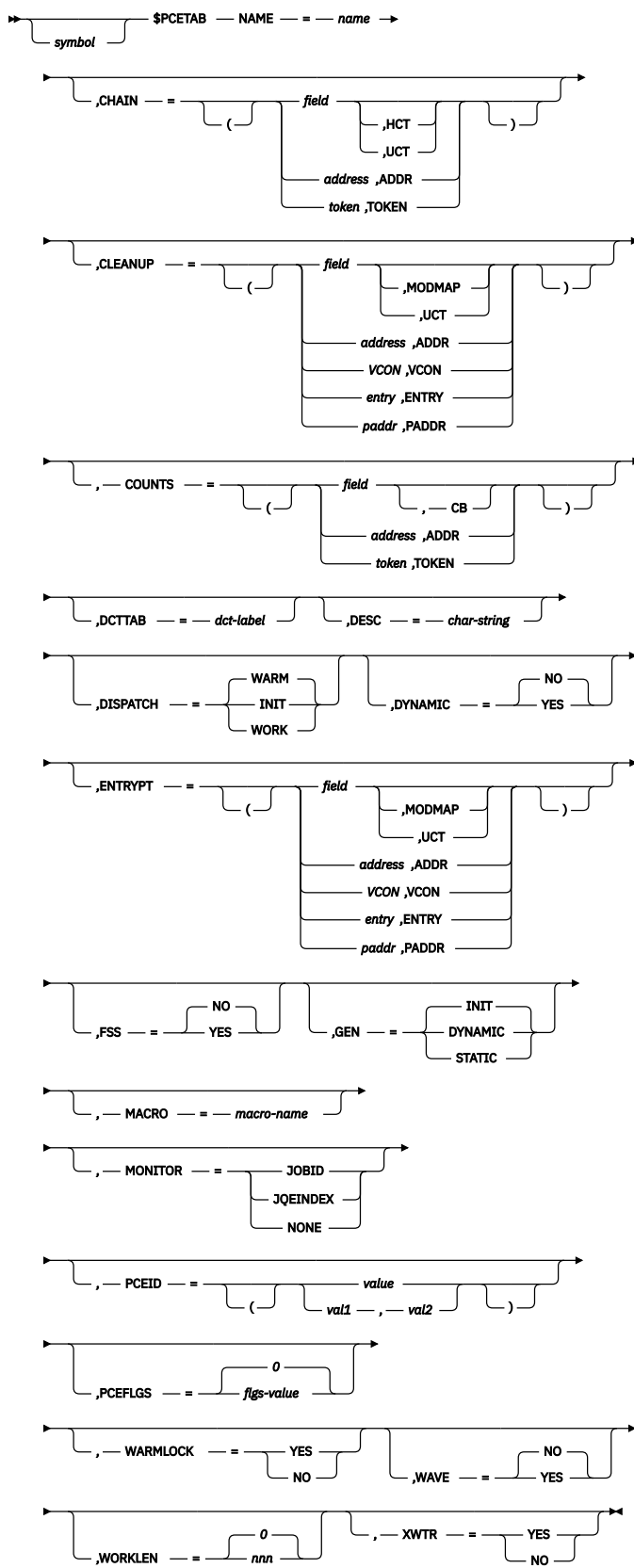
The following format description applies:

Boundary form





## Entry form



### TABLE=

Specifies the first entry in the HASP or user PCE table. If `TABLE=` is specified, all other operands are ignored. `TABLE=END` specifies the end of a PCE table; this must be coded at the end of all PCE tables.

**HASP**

Specifies the first entry in a HASP table.

**USER**

Specifies the first entry in a user PCE table.

**(USER,NOENTRY)**

Specifies that the ENTRY statement normally generated for the user PCE table is suppressed.

**DYNAMIC**

Specifies that this is a dynamic PCE table.

**END**

Specifies the end of the defined USER or HASP table.

**NAME=**

Specifies a 1- to 8-character name for the PCE type.

**CHAIN=**

Specifies the name of a fullword field that can be used to point to the first PCE of this type within the entire PCE chain that is anchored in \$PCEORG in the PCE. The \$PCEDYN service maintains this field.

**field**

Specifies an HCT field.

**(field,HCT)**

Explicitly specifies an HCT field.

**(field,UCT)**

Explicitly specifies a UCT field.

**(label,ADDR)**

Specifies the label of a field in the current CSECT.

**(offset,TOKEN)**

Specifies that the field is at the specified offset in a control block pointed to by a name/token pair. The CHAINTK parameter specifies the name and level of the token.

**CHAINTK=**

Specifies the NAME associated with a name/token pair (created using the \$TOKENSR service) which contains the address of the control block that contains the CHAIN pointer. NAMES can be up to 16 bytes long, and it must match the name specified on a \$TOKENSR call. The first 4 bytes of the token are assumed to be the control block address. The remaining 12 bytes of the token are not used.

The control block address (from the token) is combined with the offset specified on CHAIN= to determine the chaining field. CHAINTK= is required, and only allowed, if CHAIN=(offset,TOKEN) was specified.

NAME/TOKEN pairs can be created at various levels. JES2 supports tokens at the SYSTEM, HOME or TASK level. The second operand on CHAINTK= specifies the level passed on the \$TOKENSR service call.

**CHAINTK=(name,SYSTEM)**

Indicates SYSTEM level.

**CHAINTK=(name,SUBSYS)**

Indicates SYSTEM level with the last 4 bytes of the 16-byte name replaced by the subsystem id.

**CHAINTK=(name,HOME)**

Indicates HOME level.

**CHAINTK=(name,PRIMARY)**

Indicates PRIMARY.

**CHAINTK=(name,TASK)**

Indicates TASK level.

**CHAINTK=name**

Defaults to TASK level.

**CLEANUP=**

Specifies a routine to gain control if the PCE is detached with the \$PCEDYN DETACH service. This routine will not be called if the PCE is also ENDED.

**field**

Specifies a MODMAP field if this is a HASP table, or a UCT field if this is a USER table.

**(field,MODMAP)**

Explicitly specifies a MODMAP field.

**(field,UCT)**

Explicitly specifies a UCT field.

**(label,ADDR)**

Specifies a label in the current module.

**(label,VCON)**

Specifies a label in an external module link-edited with the current module.

**(label,ENTRY)**

Specifies a label on the \$ENTRY statement for the routine.

**(label,PADDR)**

Specifies a routine whose address is in the \$PADDR.

**COUNTS=**

Specifies the name of a fullword field that contains two halfword counts for this PCE type. The first count is the count of defined PCEs, and must be set during JES2 initialization (before invoking Exit 24). The second count is the count of allocated PCEs; this count is maintained by the \$PCEDYN service.

**field**

Specifies an HCT field if this is the HASP table or a UCT field if the USER table.

**(field,HCT)**

Specifies an HCT field.

**(field,UCT)**

Specifies a UCT field.

**(label,ADDR)**

Specifies a label in the current module.

**(offset,TOKEN)**

Indicates COUNTS field CB address is in MVS name/token pair. COUNTTK= must be specified.

**COUNTTK=**

Specifies the NAME associated with a name/token pair (created using the IEANTCR callable service) which contains the address of the control block that contains the CHAIN pointer. NAMES can be up to 16 bytes long (and must match the name specified on a IEANTCR call). The first 4 bytes of the token are assumed to be the control block address. The remaining 12 bytes of the token are not used.

The control block address (from the token) is combined with the offset specified on COUNT= to determine the chaining field. COUNTTK= is required, and only allowed, if COUNT=(offset,TOKEN) was specified.

NAME/TOKEN pairs can be created at various levels. JES2 supports tokens at the SYSTEM, HOME, or TASK level. The second operand on COUNTTK= specifies the level passed on the IEANTRT service call.

**COUNTTK=(name,SYSTEM)**

Indicates SYSTEM level

**COUNTTK=(name,SUBSYS)**

Indicates SYSTEM level with the last 4 bytes of the 16-byte name replaced by the subsystem id.

**COUNTTK=(name,HOME)**

Indicates HOME level

**COUNTTK=(name,PRIMARY)**

Indicates PRIMARY

**COUNTTK=(name,TASK)**

Indicates TASK level

**COUNTTK=name**

Defaults to TASK level

**DCTTAB=**

Specifies the label provided on the DCT table entry that corresponds to the DCTs that have a one to one correspondence with the PCE type (if any).

**DESC=**

Specifies a 1- to 32-character description of the PCE type. The word processor is appended to the end of this description.

**DISPTCH=**

Specifies the initial dispatching status for PCEs of this type once they are created.

**WARM**

Specifies that PCEs are dispatched immediately if initialization and warm-start processing have completed, otherwise, the PCEs are \$WAITed on HOLD. At the end of warm-start processing, all PCEs are POSTed for HOLD.

**INIT**

Specifies that PCEs are made ready immediately then dispatched at the completion of initialization processing (at the same time that warm start processing begins).

**WORK**

Specifies that PCEs are \$WAITed on work until \$POSTed by later processing.

**DYNAMIC=**

Specifies whether (YES) or not (NO) this PCE can be attached or deleted using a \$PCEDYN service call. DYNAMIC=YES is meaningful only on a \$PCETAB GEN=INIT call.

**ENTRYPT=**

Specifies the name of a fullword field containing the processor entry point address.

**field**

Specifies a MODMAP field if this is a HASP table or a UCT field if this is a user table.

**(field,MODMAP)**

Explicitly specifies a MODMAP field.

**(field,UCT)**

Explicitly specifies a UCT field.

**(label,ADDR)**

Specifies a label in the current module.

**(label,VCON)**

Specifies a label in an external module link-edited with the current module.

**(label,ENTRY)**

Specifies a label on the \$ENTRY statement for the routine.

**(label,PADDR)**

Specifies a routine whose address is in the \$PADDR.

**FSS=**

Specifies whether this PCE type is permitted (YES) or not (NO) to run in functional subsystem (FSS) mode. If FSS=YES is specified, then the larger of the JES2-mode PCE work area or the FSS-mode work area is used. The default is NO.

**GEN=**

Specifies when this specific PCE type is to be generated by the \$PCEDYN macro.

**INIT**

Specifies that PCEs are to be generated during JES2 initialization processing (that is, after most initialization, but before calling Exit 24).

## DYNAMIC

Specifies that PCEs are to be generated by using the \$PCEDYN service after initialization.

## STATIC

Specifies that this type PCE should not be generated.

**Note:** This specification is only used for the HASP initialization PCE.

**MACRO=**

Specifies a 1- to 8-character macro name for the macro that maps the PCE work area for this PCE type. This keyword is for documentation only.

**MONITOR=**

Specifies whether to set the JOBID, JQEINDEX or nothing for the JES2 monitor. Controls what is set in PCEJOBID or PCEJQEIX.

**PCEFLGS=**

Specifies the flags to place in the PCEFLAGS field during initialization. The flag values are defined with the PCEFLAGS field in the PCE mapping macro. The default is 0.

**PCEID=**

Specifies the values for the PCEID field. The second byte of the PCEID in user table entries should start at 255 and decrease. The two-byte PCEID must uniquely define a PCE type.

**value**

Specifies the PCEID as the 2-byte value defined by a DC of AL1(0,value).

**(val1,val2)**

Specifies the PCEID as the 2-byte value defined by a DC of AL1(val1,val2).

**WARMLOCK=**

Specifies whether or not this PCE type is allowed to obtain a JOE warm start lock.

## WAVE=

Specifies whether or not to obtain a work access verification element (WAVE) during processor initialization. JES2 places the address of the WAVE in the PCEWAVE field. The default for WAVE= is NO.

**WORKLEN=**

Specifies the length of the PCE work area for this PCE type. This value defaults to 0, and is typically specified using an equate in a PCE work area mapping macro.

**XWTR=**

Specifies whether or not this PCE type is a PCE that services PSO or SAPI external applications.

## Environment

- \$WAIT is not applicable – this macro generates a DSECT or a static table entry; it does not generate executable code.

## \$PCETERM – Terminate a processor control element (PCE)

Use \$PCETERM to allow a PCE to \$WAIT on the PCETM resource queue until the MISC processor detaches the PCE. This will cause the PCE that issues the macro to be terminated. If the PCE is mistakenly \$POSTed, the macro will reissue the \$WAIT PCETM until the PCE is detached.

**Note:** Control is never returned to the caller from this macro.

### Format description



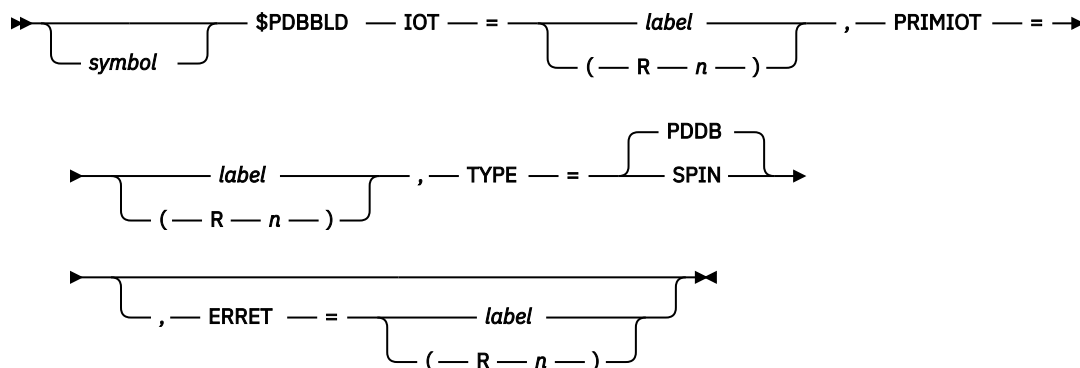
## Environment

- JES2 main task.
- \$WAIT until PCE detached.

## \$PDBBLD – Build a peripheral data definition block (PDDB)

Use \$PDBBLD to build a peripheral data definition block (PDDB). \$PDBBLD obtains a PDDB slot in the input/output table (IOT) and initializes it. If the specified IOT is not large enough to contain the new PDDB, \$PDBBLD obtains an additional IOT. On return to the caller, the address of the PDDB is in R1.

## Format description



### IOT=

Specifies a label or register that contains the address of the input/output table (IOT)

### PRIMIOT=

Specifies a label or a register that contains the address of the primary allocation IOT that is used if TYPE=SPIN or if an additional PDDB IOT is required.

### TYPE=

Specifies the type of PDDB to be built.

### PDDB

Indicates a request for a PDDB (in the specified IOT). If the IOT is not sufficiently large to contain the new PDDB, a new IOT is also created. This is the default.

### SPIN

Indicates a request to create a SPIN IOT.

### ERRET=

Specifies a label or a register that contains the address of the routine that is to receive control if \$PDBBLD does not successfully complete processing.

## Return codes

The following return codes (in decimal) are returned in register 15.

### Return Code

#### Meaning

0

PDDB successfully obtained; register 1 contains the address of the PDDB.

4

PDDB not obtained; GETMAIN for IOT failed.







**Note:**

1. Paging is done synchronously; that is, on return from \$PGSRVC the paging action is complete, and no other JES2 processor receives control during this processing.
2. For page-free requests, the page is made pageable only when the number of page-free requests specifying the page equals the number of page-fix requests for the page. If the designated area is not page-fixed, the area is unaffected by the execution of this macro instruction.
3. For page-release operations, if the area does not encompass one or more complete pages, the area is unaffected by the execution of this macro instruction.
4. Register 15 is used to pass control to the specified service routine.
5. See *z/OS MVS Programming: Authorized Assembler Services Guide* for further information concerning virtual page services.

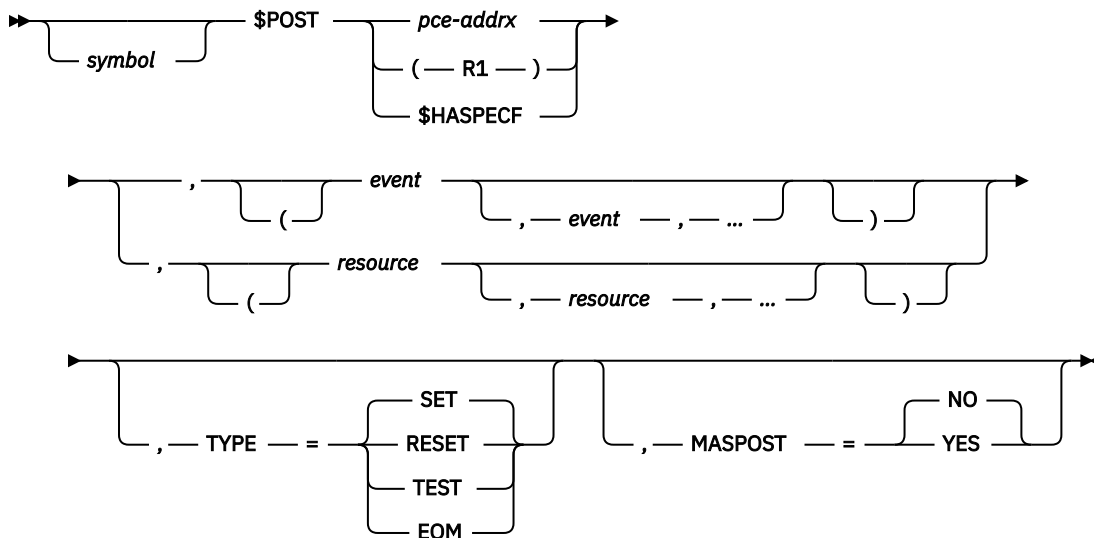
## Environment

- Main task or during JES2 initialization and termination.
- \$WAIT cannot occur.

## \$POST – Post a JES2 event complete

Use \$POST to indicate that one or more JES2 resources should be posted by turning off specified inhibitors in the \$HASPECF field of the HASP communications table (HCT). Use \$POST also to post a specific PCE that a JES2 event has occurred; if all inhibitors are reset by the action, the PCE is requeued to the JES2 dispatcher's \$READY queue. Inhibitors turned off in the \$HASPECF field cause requeuing of all PCEs on the resource wait queues by the dispatcher.

## Format description



**pce**

Specifies the specific processor control element (PCE) that is to be posted or specifies that the \$HASPECF field within the HASP communication table (HCT) is to be posted. If register 1 is used, register 1 must refer to a PCE and must be loaded with the address of the PCE before executing the macro instruction. This is a positional operand and must be specified first.

**event/resource**

Specifies one or more events/resources that are to be posted. You can specify multiple events or resources, but not a combination of events and resources on a single \$POST macro call. This operand must be consistent with the allowable events acceptable for the first operand as follows.

## **\$POST**

If PCE was specified, the following JES2 events can be specified:

### **null**

The specified PCE is made ready for dispatching if it has no wait flags on (a \$WAIT with INHIBIT=NO is issued).

### **FORCE**

The specified PCE is made ready for dispatching regardless of its wait flags.

### **HOLD**

An operator has entered a \$\$ command.

### **IO**

An input/output operation has completed (logically).

### **OPER**

An operator has started a processor. (There are no posts (\$POST) with OPER in the distributed system.)

### **POST**

An MVS POST of an ECB has been performed.

### **WORK**

Work is available for the specified processor.

If \$HASPECF was specified, the following JES2 resource can be specified:

### **ABIT**

Waiting for the next dispatcher cycle.

### **ALICE**

PCEs waiting for incomplete warm start.

### **ALOC**

A dynamic allocation has completed.

### **ARMS**

Automatic restart manager support services.

### **BERTL**

Waiting for a BERT lock to free.

### **BERTW**

Waiting for a free BERT.

### **BREG**

PCEs waiting for WLM registration requests.

### **BUF**

A JES2 buffer has been released.

### **CCAN**

Cancel JOB/TSU/STC in conversion.

### **CKPT**

A JES2 checkpoint write has completed.

### **CKPTL**

Looking for CKPT READ.

### **CKPTP**

A checkpoint cycle has completed.

### **CKPTW**

A JES2 checkpoint should be written.

### **CMB**

A console message buffer has been released.

### **CNVT**

A converter has been released.

**DAWN**

Post PCEs waiting for work notifications.

**DILBERT**

Waiting for \$DILBERT requests.

**FSS**

A functional subsystem has completed FSS-level processing.

**EDSQ**

Email Delivery Services queue has changed status.

**GENL**

Provides a method of communication from one processor control element (PCE) to another. It does not provide serialization between the PCEs. You must ensure that the condition of the waiting PCE is satisfied before it is posted. Frequent use of the GENL resource name has a severe impact on your installation's performance.

**HOMOG**

PCEs waiting for JESplex version change.

**HOPE**

An output processor has been released.

**IMAGE**

A UCS or FCB image has been loaded.

**IRCLEAN**

Internal reader cleanup needed.

**JCMD**

A JES2 job queue element has been marked for cancel (\$C) or restart (\$E) processing.

**JOE**

A JOE has been released.

**JOEI**

JOE index services.

**JOT**

A JES2 job output element has changed status.

**JOB**

A JES2 job queue element has changed status.

**JQRB**

Process JQRB queue.

**LOCK**

A lock has been released.

**MAIN**

Storage is available.

**MLLM**

Line manager resource \$POSTs.

**MFMT**

PCEs waiting for SPOOL mini-format conversion.

**NEWS**

PCEs waiting for a JNEW update (part of JESNEWS process).

**NRM**

PCEs checking the availability of NJE devices and initiating their starts appropriately.

**PCETM**

Waiting for resource manager to detach PCE.

**PSO**

A process SYSOUT request has been queued for the JES2 PSO processor(s).

## **\$POST**

### **PURGE**

A JES2 job queue element (JQE) has been placed on the purge queue.

### **PURGS**

Purge resources from \$PURGER have been released.

### **RMWT**

Resource manager has finished work.

### **RSV**

A JES2 RESERVE has been satisfied.

### **SMF**

AN SMF buffer has been released.

### **SPI**

PCEs waiting for SYSOUT API requests.

### **SPIN**

A spin data set has been created.

### **STAC**

STATUS/CANCEL resource type.

### **TRACK**

A track group from the JES2 spooling data set has been released.

### **TRACP**

Waiting for a spool track group for a privileged job.

### **UNIT**

A device control table has been released.

### **WARM**

A member of the MAS needs to be warm-started.

### **XMITJOB**

A JES2 job queue element (JQE) has been placed on the XMIT queue to be transmitted to another node.

### **TYPE=**

Specifies the type of action for a \$POST of a resource. This keyword is ignored for PCE \$POSTs of events, FORCE, or null. JES2 returns a nonzero condition code if the resource/event is POSTed, or a condition code of 0 if it is not POSTed.

### **SET**

Indicates that the resource should be POSTed (that is, the resource flag(s) set on).

### **RESET**

Indicates that the resource should be unPOSTed (that is, the resource flag(s) set off).

### **TEST**

Indicates that the resource should be tested to determine if it was \$POSTed. TYPE=TEST and MASPOST= are mutually exclusive.

### **EOM**

Post PCEs waiting for End Of Memory events.

### **MASPOST=**

Specifies whether (YES) or not (NO) the resource \$POST is to be propagated to all members of the multi-access spool complex. This keyword is only valid for resources within the JES2 main task, for example, JOB, CNVT, SPIN, HOPE, PURGE, JOE, etc. MASPOST= and TYPE=TEST are mutually exclusive.

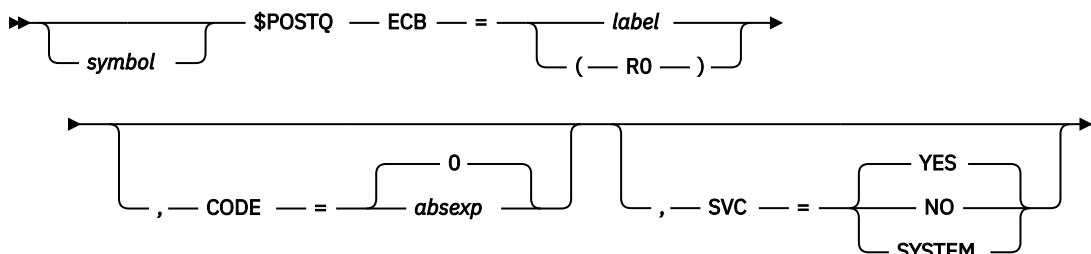
## **Environment**

- Main task.
- \$WAIT cannot occur.

## \$POSTQ – Quick post facility

Use the \$POSTQ macro to quick post an ECB (event control block). This macro produces the necessary inline code to either quick post (that is, bypass the POST routine) an ECB and/or issue an MVS POST if the specified ECB is currently waiting on an event.

### Format description



#### ECB=

Specifies the address of the ECB to be quick posted. If the ECB is currently waiting and you also specify SVC=YES on this macro, JES2 then requests that an MVS POST of the ECB be issued. If this keyword is not specified, an assembly error will occur.

#### CODE=

Specifies a 30-bit post code to be quick posted into the ECB. The default is 0.

#### SVC=

Specifies whether (YES) or not (NO) an MVS POST should be issued if the ECB is currently waiting (that is, the ECB wait bit is on). If SVC=NO is specified, no MVS POST is issued and a condition code is returned to the caller to signify whether the quick post was successful (CC=0) or unsuccessful (CC=1). The default is YES.

SVC= supports the new SYSTEM value. SVC=SYSTEM generates an MVS POST with LINKAGE=SYSTEM. SVC=YES generates an MVS POST with no other operands.

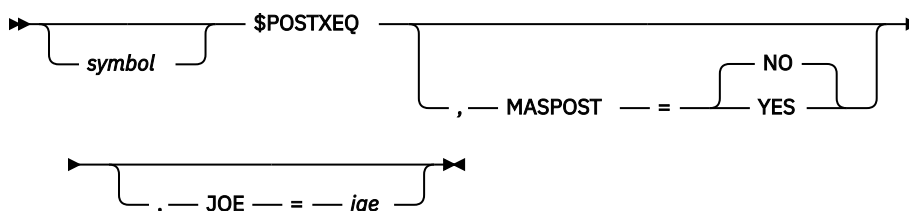
### Environment

- JES2 main task, subtask, user address space, and HASPFSSM address space.
- \$WAIT cannot occur.

## \$POSTXEQ – Wake up the EXECUTION PCE

The \$POSTXEQ macro wakes up the execution processor in an appropriate manner, so that the post is not missed if the processor is waiting somewhere other than at the top of the loop.

### Format description



#### MASPOST=

Specifies whether the execution PCEs are to be posted on other members of the MAS. The default is MASPOST=NO.

**JQE=**

Specifies a JQE to be used as an indicator that members need POSTing for.

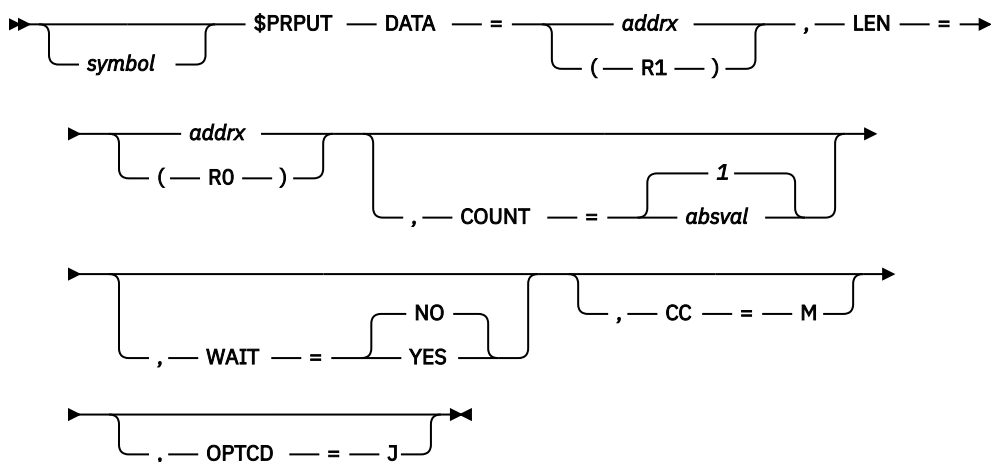
## Environment and Serialization

- This macro only executes in the JES2 main task.
- No serialization is required.

## \$PRPUT – Create separator pages

Use \$PRPUT to create user-defined separator page(s). The created separator page can replace or add to the standard separator page. The separator page JES2 Exit 1 is in module HASPPRPU.

## Format description

**DATA=**

Specifies the actual address of the data to be printed or punched. The address of this data must not be a 31-bit address. If you do specify a 31-bit address, unpredictable results may occur. The data pointed to by this register must be a fixed-data field because this data area is the I/O data area. Therefore, if \$GETBUF is used to obtain this area, be certain to add FIX=YES to that macro statement.

**LEN=**

Specifies the actual length of the fixed-data field, including any carriage control and 3800 table reference characters (TRC) if present.

**COUNT=**

Specifies the number of times the data is to be produced. Default is no repetitions.

**WAIT=**

Specifies whether to wait until I/O has completed as follows:

**YES**

Wait for I/O completion.

**NO (default)**

Do not wait for I/O to complete.

**CC=M**

Specifies that an installation-specified machine carriage control is required. The carriage control characters are defined as hexadecimal values. For the device-dependent printer control commands, refer to the “Commands” section of the manual of the printer on which you are printing your output. If this keyword is omitted, no carriage control is assumed.

**OPTCD=J**

Specifies that the 3800 table reference character (TRC) is present in the data. If this parameter is omitted, 3800 TRC is not assumed to be present.

## Return codes

The following return codes (in decimal) are returned in register 15:

Return Code	Meaning
0	Successful creation of the separator page
4	Creation of the separator page was suspended or terminated.

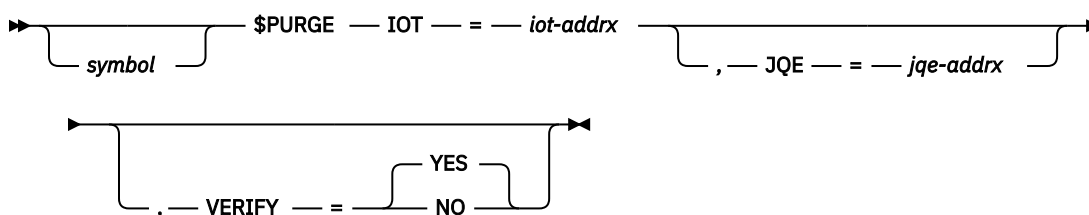
## Environment

- Main task.
- \$WAIT can occur if WAIT=YES is specified.

## \$PURGE – Return direct-access space

Use \$PURGE to return the direct-access space that was allocated for a given job or data set.

## Format description



### IOT=

Specifies the address of the primary allocation IOT from which track group allocation elements (TGAEs) are to be returned. Secondary allocation IOTs, if any exist, are also processed.

### JQE=

Specifies the address of a JQE. The job-key field of the JQE must match the job-key field of the specified IOT. Use of the JQE parameter causes JES2 to perform additional validation to ensure that the space returned belongs to the job identified in the JQE.

- If the job-key fields match and the IOT Parameter points to an IOT, JES returns the allocated space to the system.
- If the job-key fields do not match, JES does not return the allocated space and issues a \$DISTERR macro.

### VERIFY=YES | NO

Specifies whether to call SAF for authorization before continuing the purge process. Use VERIFY=NO when purging buffers which do not have a profile name SAF can use to verify access authority.

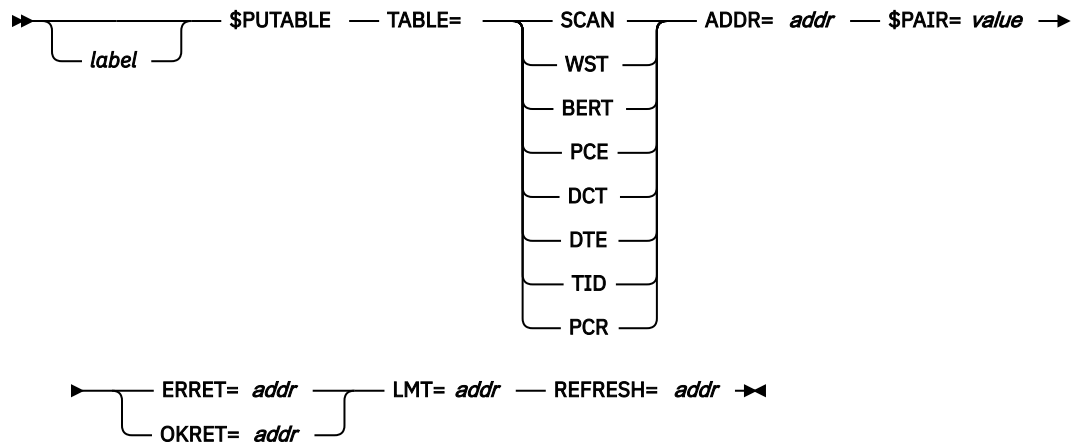
## Environment

- Main task.
- \$WAIT can occur.

## \$PUTABLE – Add HASP/user table entry

Use \$PUTABLE to add a table to a MCT table pair. See [Appendix A, “Using JES2 table pairs,”](#) on page 425 for a detailed description of table pairs.

Format description



**TABLE=**  
Specifies the table pair to be accessed.

- TABLE=SCAN - \$SCANTAB
- TABLE=WST - \$WSTAB
- TABLE=BERT - \$BERTTAB
- TABLE=PCE - \$PCETAB
- TABLE=DTE - \$DTETAB
- TABLE=DCT - \$DCTTAB
- TABLE=TID - \$TIDTAB
- TABLE=PCR - \$PCTAB

**ADDR=**  
Specifies the address of the table to be linked in.

**\$PAIR=**  
Specifies the specific offset and control block of the table pair to which the dynamic table should be chained

**ERRET=**  
The address to go to if the table could not be added.

**OKRET=**  
The address to go to if the table is successfully added.

**LMT=**  
Specifies the address of the LMT of the module that contains the specified new table name.

**REFRESH=**  
Specifies the address of the table to replace with the specified new table address.

Return codes

The following return codes (in decimal) are returned in register 15:

Return Code	Meaning
0	Requested table entry was found.
4	Requested table entry was not found or end-of-table.



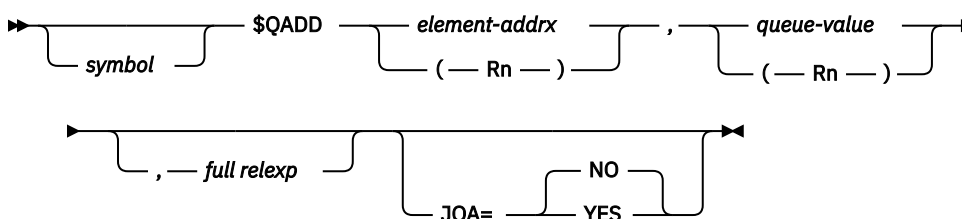
## Environment

- Main task

## \$QADD – Add job queue element to the JES2 job queue

Use \$QADD to add an element to the JES2 job queue, placing it in the specified logical queue. The address of the job queue element where the element image has been placed is returned in register 1 if the element is successfully added.

## Format description



### element

Specifies the address of an element image which is to be added to the JES2 job queue. If register notation is used, the address must be loaded into the designated register before the execution of this macro instruction.

### queue

Specifies the logical queue in which the job queue element is to be placed. This value must always be one of the eight logical queue types. If register notation is used, one of these values must be loaded into the designated register before the execution of this macro instruction.

The queue type specifications may be ignored if the job queue element has been flagged for cancellation. The resulting logical queue is as follows:

- JQE1OCAN bit on and JQE1PURG bit off. Any \$QADD with a \$XEQ or \$XMIT specification is altered to \$OUTPUT.
- JQE1OCAN bit off and JQE1PURG bit on. Any \$QADD with JQEJOECT and JQEHLDCCT fields 0 is altered to \$PURGE. Any \$QADD with a JQEJOECT or JQEHLDCCT field nonzero is altered to \$HARDCPY.

### full

Specifies a location to which control is returned if the JES2 job queue is full. If this operand is omitted, the condition code is set to reflect the status of the JES2 job queue as follows:

#### CC=0

The queue is full and the element cannot be accepted.

#### CC≠0

The element was successfully added to the queue.

### JQA=

JQA=YES indicates that a prototype JQA was passed on LOC=. \$QADD will ensure that the JQX and BERT updates are made. Default is JQA=NO.

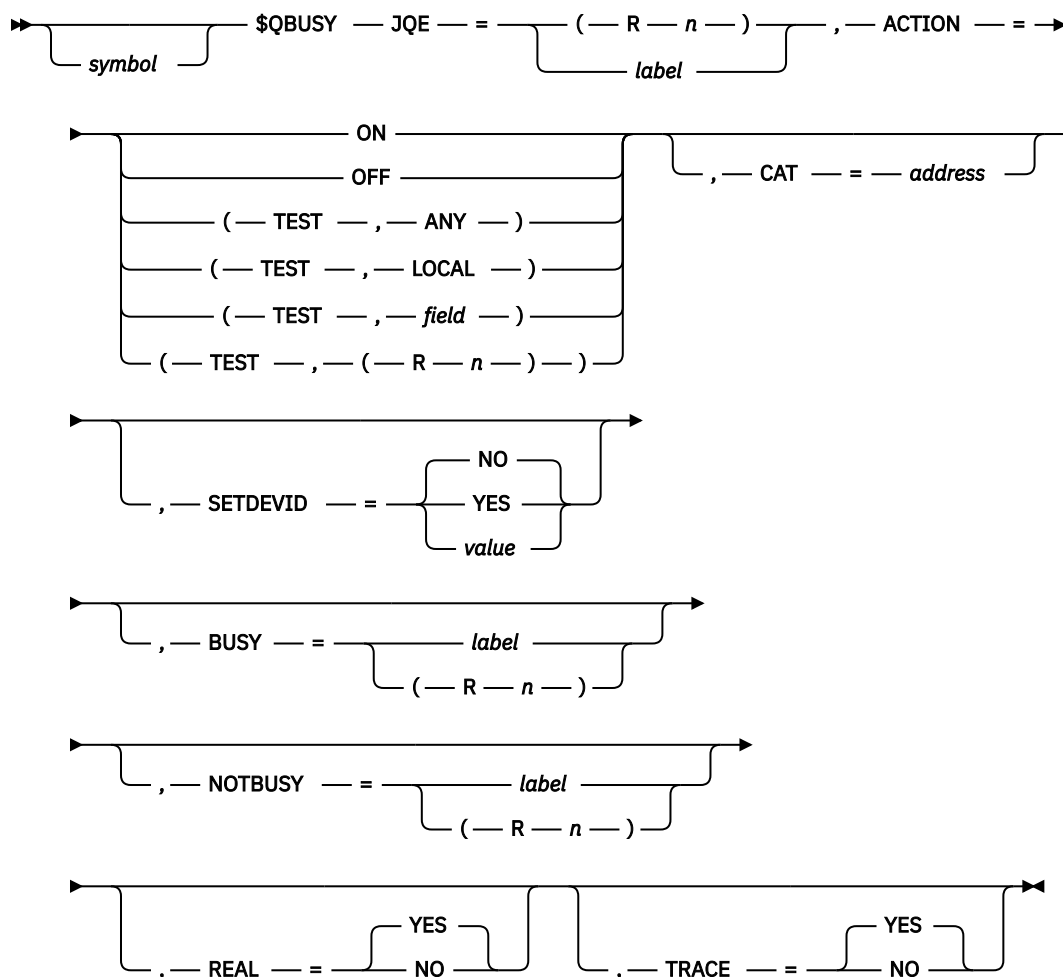
## Environment

- Main task.
- \$WAIT can occur.

## \$QBUSY – Set or test the JQE busy–system indicator

Use \$QBUSY to set or test the busy system indicator for a job queue element (JQE).

### Format description



#### JQE=

Specifies the address of the JQE whose busy indicator is to be set, reset, or tested. If you use register notation, provide the address of the JQE in the specified register. If you specify a label, that label is the address of the JQE. (For example, you can specify JQE=JQE if you set a base register for the JQE DSECT.)

#### ACTION=

Specifies whether the busy indicator for this JQE is to be set on (ON) or turned off (OFF).

##### ON

Indicates this member is processing this element.

##### OFF

Indicates that this element is not being processed by any members.

##### (TEST,ANY)

Indicates that the JQE should be tested to determine if the JQE is busy on any member of the MAS.

##### (TEST,LOCAL)

Indicates that the JQE should be tested to determine if the JQE is busy on this member of the MAS.

**(TEST,field)**

Indicates that the JQE should be tested to determine if the JQE is busy on the member of the MAS whose member number is specified in the indicated 1-byte field.

**(TEST,(Rn))**

Indicates that the JQE should be tested to determine if the JQE is busy on the member of the MAS whose member number is specified in the indicated register.

**Note:** Whenever setting ACTION=(TEST,...), you must also specify either BUSY= or NOTBUSY= to indicate to JES2 where it should branch based on the test result.

**CAT=**

Specifies the address of the update mode CAT. If the caller has an update mode CAT associated with the class the job is currently active in, the address must be passed to \$QBUSY.

**SETDEVID=value|YES|NO**

Specifies whether (YES) or not (NO) the device id in the JQE is to be set. If a value is specified, it will be set to JQEDEVID. SETDEVID=NO is the default.

**BUSY=**

Specifies a label or register to which to branch if the JQE is busy on the particular member of the MAS.

BUSY= is only valid if you also specify ACTION=(TEST,...).

**NOTBUSY=**

Specifies a label or register to which to branch if the JQE is **not** busy on the particular member of the MAS.

BUSY= is only valid if you also specify ACTION=(TEST,...).

**REAL=**

Specifies whether this JQE is a 'real' JQE within the JES2 checkpoint data set (YES) or a prototype JQE in a work area (NO). If REAL=YES, JES2 validates the JQE.

TRACE=YES and REAL=NO are mutually exclusive.

**TRACE=**

Specifies whether (YES) or not (NO) this modification to the busy indicator is to be traced by the SYSjes2 component trace. See *z/OS MVS Diagnosis: Tools and Service Aids* for further information concerning SYSjes2 component tracing.

**YES**

Indicates that tracing is set on for this \$QBUSY call.

TRACE=YES and REAL=NO are mutually exclusive.

**NO**

Indicates that tracing is set off for this \$QBUSY call.

**Note:**

1. TRACE= is only valid if you also specify either ACTION=ON or ACTION=OFF.
2. IBM suggests that you do not turn SYSjes2 tracing off. If JES2 encounters a problem related to \$QBUSY services, the data obtained from this trace can significantly aid debugging procedures.

## Environment

- Main task.
- \$WAIT or WAIT cannot occur.

**Note:**

1. On return from the \$QBUSY routine, register 15 will contain a 0 (zero) if you specified ACTION=ON or ACTION=OFF. JES2 provides no return codes for ACTION=(TEST,...).
2. Register usage
  - \$QBUSY uses registers R0, R1, R14, and R15 as work registers.

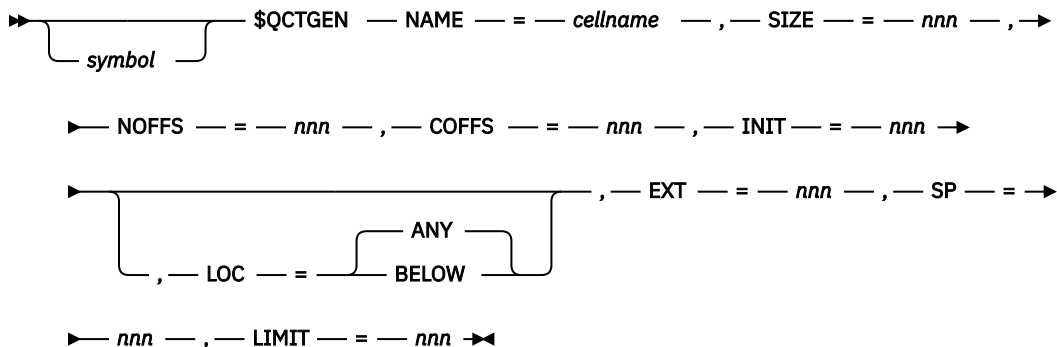
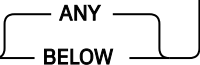
## \$QCTGEN

- On entry, \$QBUSY requires that R11 contain the address of the HCT.

## \$QCTGEN – Define a quick cell control table

Use \$QCTGEN to define the attributes of a quick cell type in a quick cell control table (QCT). Note that **only** **LOC=** is optional.

### Format description

→   
\$QCTGEN NAME = *cellname* , SIZE = *nnn* ,  
NOFFS = *nnn* , COFFS = *nnn* , INIT = *nnn* ,  
EXT = *nnn* , SP = ,  
LOC =  ,  
LIMIT = *nnn* →

#### NAME=

Specifies (in EBCDIC) the name of the quick cell control table.

#### SIZE=

Specifies the size (in bytes) of an individual quick cell.

#### NOFFS=

Specifies the offset of the NAME field in the quick cell.

#### COFFS=

Specifies the offset of the CHAIN field in the quick cell.

#### INIT=

Specifies the number (0-32767) of quick cells created in the initial quick cell pool.

#### EXT=

Specifies the number (0-32767) of quick cells in a quick cell pool extension.

#### SP=

Specifies the storage subpool number where the quick cell pool resides.

#### LIMIT=

Specifies the maximum number (0-32767) of quick cells to GET/FREE at any one time.

### Environment

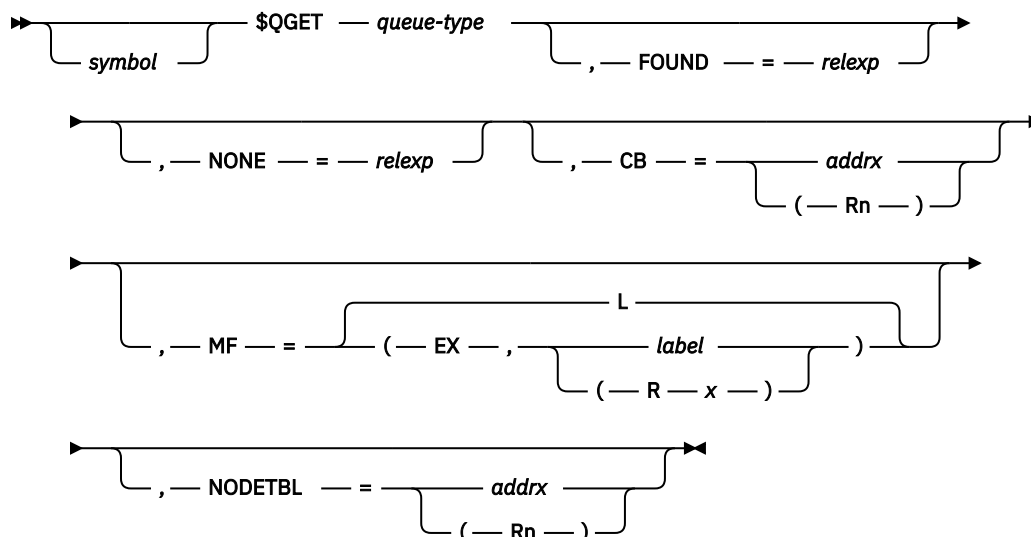
- Functional subsystem (HASPFSM).
- MVS WAIT and \$WAIT not applicable.

## \$QGET – Obtain a job queue element from the JES2 job queue

Use \$QGET to obtain a job queue element from the specified logical queue of the JES2 job queue and return the address of this element in register 1.

**Note:** JES2 returns an artificial JQE (JQA).

## Format description



### queue type

Specifies the logical queue from which the job queue element is to be obtained. This queue type is indicated in the inline parameter list generated by the macro expansion. Valid queue types and their meanings are:

#### **\$DUMMY**

Reserved queue.

#### **\$FREE**

Indicates that the JQE is to be obtained from the JES2 free queue.

#### **\$HARDCPY**

Indicates that the JQE is to be obtained from the JES2 hardcopy queue.

#### **\$INPUT**

Indicates that the JQE is to be obtained from the JES2 input queue.

#### **\$INWS**

Indicates a QGET call for JES-managed initiators.

#### **\$OJTWS**

Indicates that the JQE work selection algorithms are used to select an eligible job for this transmitter.

#### **\$OJTWSC**

Indicates that the JQE work selection algorithms are used to select an eligible job for this transmitter and that the conversion queue (`$XEQ`) is scanned for work.

#### **\$OUTPUT**

Indicates that the JQE is to be obtained from the JES2 output queue.

#### **\$PURGE**

Indicates that the JQE is to be obtained from the JES2 purge queue.

#### **\$RECEIVE**

Indicates that the JQE is to be obtained from the JES2 SYSOUT receive queue.

#### **\$SETUP**

Indicates that the JQE is to be obtained from the JES2 setup queue.

#### **\$WLMINWS**

Indicates a QGET call for WLM-managed initiators.

#### **\$XEQ**

Indicates that the JQE is to be obtained from the conversion queue.

**\$XEQCLAS**

Indicates that the JQE is to be obtained from the JES2 execution class queue.

**\$XMIT**

Indicates that the JQE is to be obtained from the JES2 transmit queue.

**Note:** Although \$INWS, \$OJTWS, and \$OJTWSC are not actual queue types, they can be used to indicate work selection for offload job transmitters or a call for initiators.

**FOUND=**

Specifies a label or address in a register to which JES2 branches if a selectable JQE is found.

**NONE=**

Specifies a label or address in a register to which JES2 branches if no selectable JQE is found.

**CB=**

Specifies the address of a control block which is to be used for work selection or an initiator call. This keyword is only valid if either the queue type is specified as \$INWS, \$OJTWS or \$OJTWSC.

**MF=**

Specifies the required form of this macro.

**L**

Indicates the list form of the macro.

**EX**

Indicates the executable form of the macro. This form requires the address of a parameter list.

**label**

The label of the parameter list.

**(Rx)**

The register that contains the address of the parameter list.

**NODETBL=**

Specifies the location of the MDCTNODS field in the job transmitter's line device control table (DCT). If the queue type specification is not \$XMIT, this keyword should not be used. If register notation is used, the address must have been loaded into the designated register before execution of this macro instruction.

## Environment

- Main task.
- \$WAIT can occur.

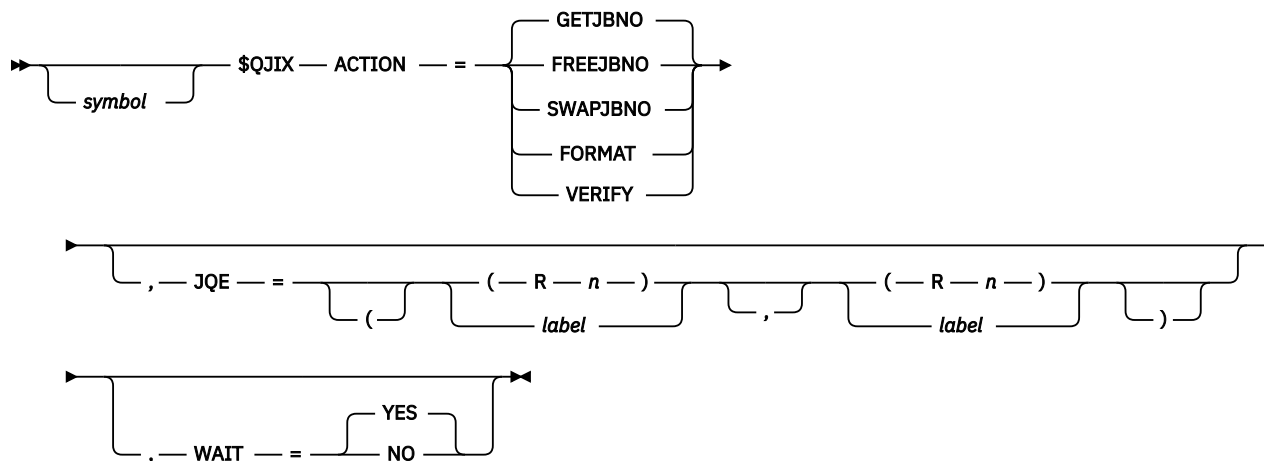
## \$QJIX – JES2 job number services

---

Use \$QJIX to allocate and deallocate JES2 job numbers, to maintain the JES2 job number table (JNT), and to maintain the job number index table (JIX). Specifically, you can use this macro to provide the following functions:

- Allocate a job number for a specified JQE
- Deallocate a job number for a specified JQE
- Swap the job numbers for two jobs as required for the “move job” function of dynamic spool volume support
- Format the JNT and the JIX during system-wide warm start and cold start processing
- Verify a JIX during an all-system warm start

## Format description



### ACTION=

Specifies the action the \$QJIX is to perform for the caller. Valid specifications are:

#### GETJBNO

Indicates that \$QJIX is to allocate a job number for the JQE provided. If a job number is unavailable, \$QJIX will \$WAIT until a number is available if WAIT=YES is coded or allowed to default. If this job is from another member, QJIX will attempt to get the original job number. If that number is unavailable, \$QJIX assigns the next available job number. ACTION=GETJBNO is the default.

#### FREEJBNO

Indicates the job number is to be freed.

#### SWAPJBNO

Indicates that the job numbers assigned to the two specified JQEs are to be swapped and the JIX is updated to reflect that change.

#### FORMAT

Indicates the job number table (JNT) and the job queue index (JIX) are to be initialized. They are formatted at initialization even if no jobs are currently in the system. Following FORMAT processing, \$QJIX will also provide VERIFY processing as described below. **This is valid only during initialization.**

#### VERIFY

Indicates that the job number of the JQE provided needs to be verified as unique and not a duplicate of any other job on the job queues. If the job number is unique, the JIX is updated; if the job number is not unique, the caller receives an error indication and the JQE is freed. Use this specification only for JES2 initialization processing.

#### JQE=

Specifies the address of the JQE to be added to the JIX. This keyword is required for all ACTION= specifications except ACTION=FORMAT. If you specify ACTION=SWAPJBNO, two JQEs are required. This can be any combination of registers or labels.

#### WAIT=

Specifies whether (YES) or not (NO) the \$QJIX routine is to \$WAIT for an available job number. WAIT= is valid only if you also specify, or allow ACTION= to default to, GETJBNO.

## Return codes

The following return codes (in decimal) are returned in register 15.

Return Code	Meaning
-------------	---------

## **\$QJQE**

**0**

Processing successful. If ACTION=VERIFY, then the job number is not already in use.

**4**

No job numbers are currently available, or if ACTION=VERIFY, then the job number was in use.

**8**

ACTION=FORMAT processing was unsuccessful. The FORMAT request occurred after JES2 initialization.

## **Environment**

- Main task and limited main task (initialization).
- \$WAIT can occur if WAIT=YES is specified.

## **\$QJQE – Obtain address of JQE queue head**

---

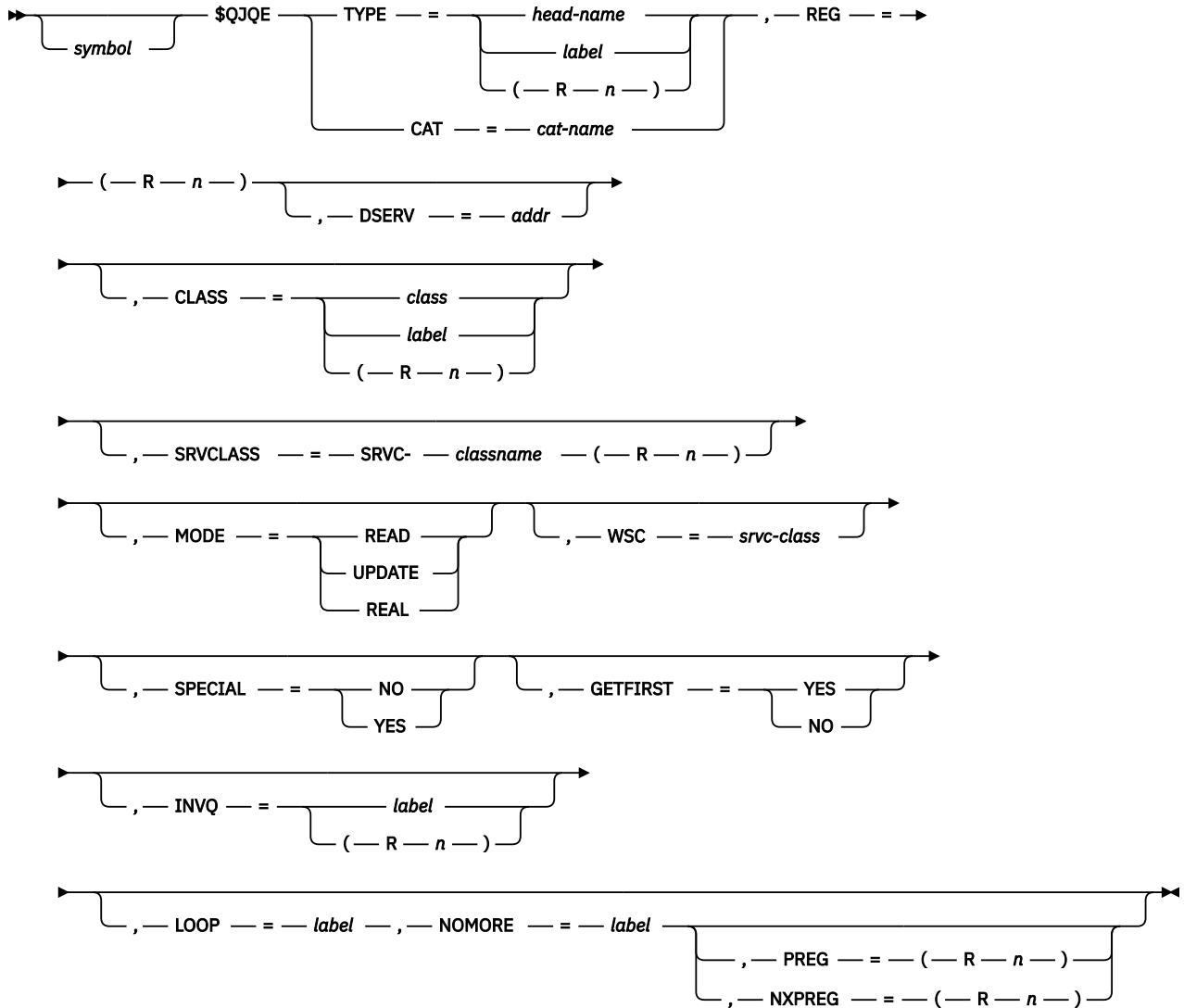
Use \$QJQE to generate inline code that finds a specified JQE queue head. JES2 searches either the HASP communication table (HCT) or a checkpoint version.

### **Note:**

1. An artificial JQE is returned.
2. \$QJQE macro expands into \$DOGCAT and \$DOGJQE services, so you need to follow the rules that apply to these macros to determine if resources should be freed with \$DOGJQE ACTION=RETURN. If a JQE is passed to the code, the application is responsible for freeing the resource using ACTION=RETURN. If no JQE was passed, there is no need to free resources. Likewise, if a CAT is being returned to your code, you need to free the resource using a \$DOGCAT ACTION=RETURN. If no CAT is returned (for example, end of queue is reached), you do not need to free resources.



## Format description



### TYPE=

Specifies the queue type of the requested JQE head. Specify a standard queue head name (listed below), a 1-byte field, or a register that contains the JQETYPE of the JQEs on the queue.

#### Note:

1. In Table 9 on page 307, for example, TYPE=EXEC is equivalent to specifying a field that contains \$XEQCLAS (X'7F').
2. If you specify TYPE=EXEC (in any form), you must also specify CLASS=.
3. Specify TYPE=FREE to scan the FREE JQE queue.

Table 9. \$QJQE Standard Queue Head Names		
Standard Queue Head Name	Field Equate	Value (hexadecimal)
INPUT	\$INPUT	X'20'
CNVT	\$XEQ	X'40'
SETUP	\$SETUP	X'08'
EXEC	\$XEQCLAS	X'7F'

*Table 9. \$QJQE Standard Queue Head Names (continued)*

Standard Queue Head Name	Field Equate	Value (hexadecimal)
SPIN	\$SPIN	X'80'
OUTPUT	\$OUTPUT	X'02'
HARDCPY	\$HARDCPY	X'01'
PURGE	\$PURGE	X'00'
RECEIVE	\$RECEIVE	X'04'
XMIT	\$XMIT	X'10'
FREE	\$FREE	X'FF'
REBUILD	n/a	n/a

**DSERV=**

Specifies the address of the DSERV control block of the checkpoint version JES2 will search. If you do not specify DSERV=, JES2 searches the current checkpoint data set.

DSERV is required and valid in any environment except the JES2 main task. If specified, the caller must be in AR ASC mode.

**Note:** You must obtain the IAZDSERV data area, which describes the checkpoint version, prior to invoking the \$QJQE service. Use a \$DSERV GET call to do so. See Appendix D, “Accessing checkpoint control blocks outside the JES2 main task,” on page 481 for a typical coding example.

**INVQ=**

Specifies a label to which JES2 branches if the combination of TYPE= and CLASS= parameters does not point to a valid head index. INVQ= is required if you specify a queue type on TYPE= with a nonstandard name.

**CLASS=**

Specifies the class of the queue JES2 searches. Valid classes are: A-Z, 0-9, TSU (X'E0'), STC (X'D0'), or a 1-byte field or register containing the class. CLASS= is only valid on a TYPE=EXEC call (any format).

**CAT=**

Specifies the CAT for the class whose queue head is required as returned by the \$DOGCAT service.

**SRVCLASS=**

Specifies the service class queue name (an 8-byte field) or a register containing the class to use.



**Attention:** LOOP= or GETFIRST=YES must be specified.

**WSC=**

Specifies the service class. Use the \$DOGWSCQ macro to obtain the service class queue head .

**MODE=**

Specifies the mode in which the JQE is required. Valid modes are those supported by the \$DOGJQE service (that is, READ and UPDATE). MODE=REAL indicates that the real JQE is to be located and is only valid when CLASS=, CAT=, or TYPE= is specified.

**SPECIAL=**

Specifies whether (YES) or not (NO) the JQE is to be obtained with update access. See “\$DOGJQE – Deliver or get JQE” on page 132 for more details.

**GETFIRST=**

Specifies whether (YES) or not (NO) the first JQE is to be obtained, rather than the zeroth JQE.



**Attention:** GETFIRST=NO is only allowed with TYPE=, CAT= or WSC= keywords and ONLY when LOOP= is NOT specified.

**REG=**

Specifies the register (R2-R10 or R12) into which JES2 is to place the job queue head address or the JQE address. If you do not specify LOOP=, JES2 returns the job queue head address.

If you specify LOOP=, JES2 returns the JQE address. If you specify LOOP= and the end of the specified queue was reached, JES2 returns a zero.

**LOOP=**

Specifies the label that JES2 uses to loop through all the JQEs on the specified queue. When doing so, the value in REG= must remain unchanged from the value \$QJQE previously returned. If you specify LOOP=, you must also specify NOMORE=. LOOP= is optional.

**NOMORE=**

Specifies the label to which JES2 should branch when there are no more JQEs on the specified queue.

NOMORE= is optional but if LOOP= or GETFIRST=YES is coded, NOMORE= must also be coded.

**PREG=**

Specifies a register into which the address of the JQE that pointed to the JQE returned in REG= maintained. This can be the 0th JQE and is used if the JQE that was returned is placed on a different queue and you want to continue looping where you left off. Place the value from this register into the register specified in REG= to resume scanning the queues. This operand is optional and is only valid when MODE=REAL and LOOP= is specified. Valid registers are R2–R10 and R12. PREG= is main task only. PREG and NXPREG are mutually exclusive.

**NXPREG=**

Specifies an optional work register to use for prefetch processing when LOOP= is specified. This register should remain unchanged for the duration of the loop. When specified and running on a processor that supports prefetch instruction, the next JQE in the chain is pre-fetched for read access when the macro returns the current JQE address. NXPREG and PREG are mutually exclusive.

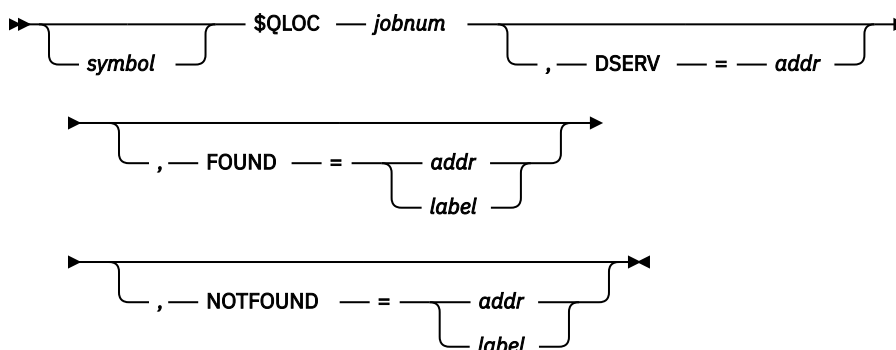
## Environment

- If you specify DSERV=, the caller can be running under any environment except the JES2 main task.
- If you do not specify DSERV=, the caller must be running under the JES2 main task.
- \$WAIT cannot occur.

## \$QLOC – Locate job queue element (JQE) for specific job

Use \$QLOC to locate the job queue element associated with the job specified by a job number and return the address of this element in register 1.

### Format description

**jobnum**

Specifies the binary job number associated with the job for which the job queue element is being searched. If an address is used, it specifies the address that contains the binary job number. If

## \$QLOCNXT

register notation is used, the binary job number must be loaded into the designated register before the execution of this macro instruction.

This is a required, positional parameter.

### DSERV=

Specifies the address of the job information service token list (MVS DSERV control block) of the checkpoint version that JES2 will search when attempting to locate the JQE. If you do not specify DSERV=, JES2 searches the current checkpoint data set.

DSERV= is only valid if the caller is in AR ASC mode.

**Note:** You must obtain the IAZDSERV data area, which describes the checkpoint version, prior to invoking the \$QLOC service. Use a \$DSERV GET call to do so. See [Appendix D, “Accessing checkpoint control blocks outside the JES2 main task,”](#) on page 481 for a typical coding example.

### FOUND=

Specifies a location to which control is to be returned if the specified job number is found in the JES2 job queue.

### NOTFOUND=

Specifies a location to which control is to be returned if the specified job number is not found in the JES2 job queue.

## Return codes

Condition codes resulting from issuing a \$QLOC follow:

Condition Code	Meaning
----------------	---------

0	The specified job was found, and R1 contains the address of the associated job queue element.
---	---

≠0	The specified job was not found.
----	----------------------------------

## Environment

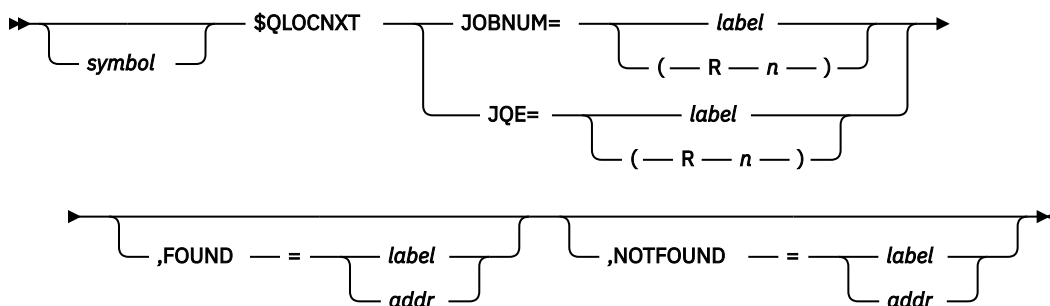
- Main task (unless DSERV= is specified)
- All but the JES2 main task (if DSERV= is specified)
- \$WAIT can occur.

## \$QLOCNXT– Find next job number after current JQE in JIX

Find the next allocated job number after the current JQE/job number in the JIX. Return the associated JQE to the caller.

If a job number of 0 is specified, the search will begin with job number 1.

## Format description



**JOBNUM=**

Current job number. JOBNUM= or JQE= are required and are mutually exclusive.

**JQE=**

Current JQE. JOBNUM= or JQE= are required and are mutually exclusive.

**FOUND=**

Specifies a label to be branched to or a register to be branched on if the next allocated job number is found.

**NOTFOUND=**

Specifies a label to be branched to or a register to be branched on if no other allocated job numbers are found after the current job number in the JIX.

## Return codes

The following return codes (in decimal) are returned in register 15:

**Return Code****Meaning****0**

Next job number found (JQE address in R1).

**4**

Next job number not found (R1 is 0).

## Environment

- Main task.

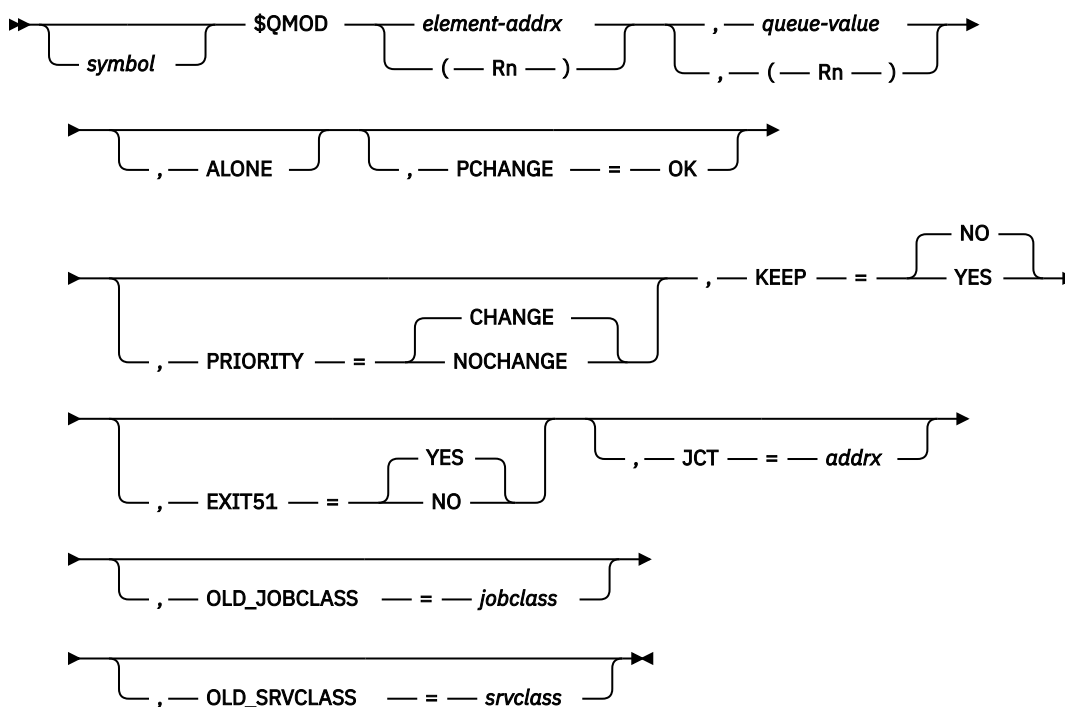
## **\$QMOD – Modify job queue element in JES2 job queue**

---

Use \$QMOD to remove a modified job queue element from the JES2 job queue and place it back on the queue in the specified logical queue according to the priority of the job queue element.

## Format description

**Note:**



## element

Specifies the address of an element that has been modified and is to be requeued in the JES2 job queue. If register 1 is used, the address must be loaded into register 1 before execution of this macro instruction.

**Note:** The JOE returned may be real or artificial.

**queue**

Specifies the logical queue where the job queue element is to be placed. This value must always be one of the eight logical queue types. If register 0 is used, one of these values must be loaded into register 0 before the execution of this macro instruction.

The queue type operands may be ignored if the job queue element has been flagged for cancellation. The resulting logical queue is as follows:

- JQE1OCAN bit on and JQE1PURG bit off. Any \$QMOD with a \$XEQ specification is altered to \$OUTPUT.
- JQE1OCAN bit off and JQE1PURG bit on. Any \$QMOD with a JQEJOECT and JQEHLDCCT field 0 is altered to \$PURGE. Any \$QMOD with a JQEJOECT or JQEHLDCCT field nonzero is altered to \$HARDCPY.



**Attention:**

If the processor issuing the \$QMOD does not have exclusive ownership of the JQE through \$QSUSE, the results of the \$QMOD macro instruction are unpredictable. One way to guarantee exclusive ownership is to obtain the JQE through a \$QGET or \$QADD macro instruction or with the \$GETLOK macro instruction.

## ALONE

Indicates that the busy flags associated with the moved element are to remain unchanged. If ALONE is not specified, the busy flags associated with the moved element are turned off.

Once the queues have been obtained, all modifications must be made to the JES2 job queues before a \$WAIT macro can be issued. Issuing a \$WAIT macro implies that the processor no longer requires the queues.



## \$QREM

- JQE10CAN bit on and JQE1PURG bit off. Any \$QPUT with a \$XEQ or \$XMIT specification is altered to \$OUTPUT.
- JQE10CAN bit off and JQE1PURG bit on. Any \$QPUT with a JQEJOECT and JQEHL DCT fields 0 is altered to \$PURGE. Any \$QPUT with a JQEJOECT or JQEHL DCT field nonzero is altered to \$HARDCPY.

### Note:

The specified job queue element must be previously obtained with a \$QGET or \$QADD macro instruction or the action of the \$QPUT macro instruction is unpredictable.

### KEEP=

Indicates a YES or NO. The option is valid only if the JQE is an artificial JQE. KEEP=NO frees the memory for the artificial JQE.

## Environment

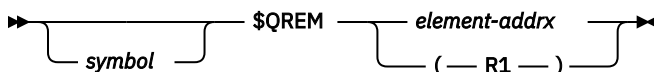
- Main task.
- \$WAIT can occur.

## \$QREM – Remove job queue element from JES2 job queue

---

Use \$QREM to remove a specified job queue element from the JES2 job queue.

## Format description



### element

Specifies the address of an element that is to be removed from the JES2 job queue. If register notation is used, the address must be loaded into the designated register before the execution of this macro instruction.

### Note:

1. The JQE returned may be real or artificial.
2. If an artificial JQE is returned, the storage is freed by the \$QREM service.



### Attention:

The specified job queue element must have been previously obtained with a \$QGET or \$QADD macro instruction or the action of the \$QREM macro instruction is unpredictable.

## Environment

- Main task.
- \$WAIT can occur.

## \$QSUSE – Synchronize to use shared queues

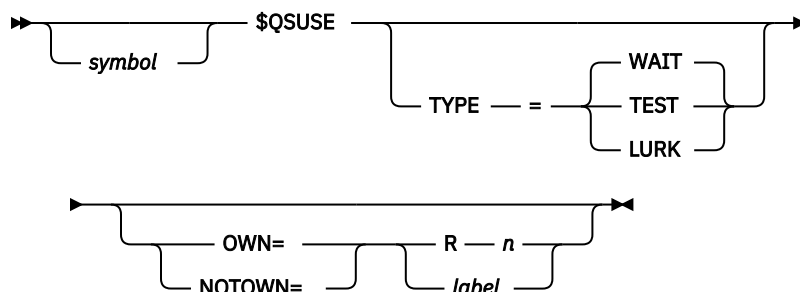
---

Any JES2 processor that begins an access to any information in the checkpoint records (which are shared if the system is a multi-access spool environment) must execute the \$QSUSE macro instruction before such access to update checkpoint records.

The contents of the checkpoint records include: shared queue control elements (QSEs), shared communications queues (SCQs), checkpointed HCT variables (beginning at \$SAVEBEG, including job queue headers), remote message spooling queues, remote sign-on table, master track group map, job queue, and job output table (JOT).



## Format description



### TYPE=

Execution of the macro tests the \$QSONDA bit and \$CKPTACT bit in the \$STATUS field of the HCT. Updating any checkpointed information is permitted only if both bits are 0. If TYPE= is not specified, WAIT is the default.

### WAIT

Indicates that the calling processor waits (\$WAIT CKPT) access to update the checkpoint records is permitted. The checkpoint processor is activated, if necessary, and it later posts (\$POST) all other processors forced to wait (\$WAIT CKPT).

OWN= and NOTOWN= are not valid if TYPE=WAIT

### TEST

Indicates that an immediate return to the caller occurs, with a zero condition code if updating is permitted.

The permission to update granted by execution of this macro expires when the processor executes any \$WAIT macro instruction, actual or imbedded in another macro.

OWN= and NOTOWN= are optional if TYPE=TEST

### LURK

Indicates that control should be returned to the caller once queue ownership is established by any PCE. This call type differs from the WAIT type because LURK does not request that JES2 interrupt checkpoint cycle processing unless other PCEs have also requested the queues.

Use TYPE=LURK to make a passive request for queue ownership. LURK can be useful for processing non-critical work, which if it ever gets done does require queue ownership. However, processing work in this manner avoids the possibility of checkpoint contention.

When control returns from a \$QSUSE TYPE=LURK call, there is no guarantee this JES2 member will own the queues.

OWN= or NOTOWN= or both must be specified if TYPE=LURK

### OWN=

Specifies a register that contains the address of the routine or label to which to branch if the queues are owned by this member.

### NOTOWN=

Specifies a register that contains the address of the routine or label to which to branch if the queues are **not** owned by this member.

## Environment

- Main task.
- \$WAIT can occur if TYPE=WAIT or TYPE=LURK is specified.

## \$QUESMFB – Queue a JES2 SMF buffer on the busy queue

Use \$QUESMFB to place a JES2 SMF buffer address on the busy queue (\$SMFBUSY) and MVS post (POST) the HASPACCT subtask.

### Format description



#### buffer-addr

Specifies the address of the buffer to be queued. If register notation is used, the buffer address must be loaded into the designated register before the execution of this macro instruction.

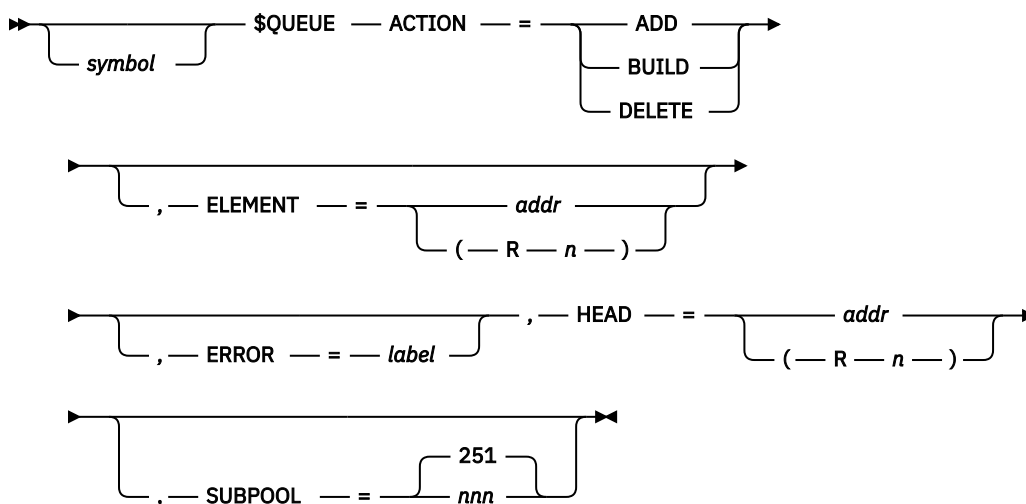
### Environment

- Main task.
- \$WAIT cannot occur.

## \$QUEUE – Maintain a first-in-first-out (FIFO) queue

Use \$QUEUE to build, add an element to, or get an element from a FIFO queue. This macro provides an easy method of maintaining queues for any purpose.

### Format description



#### ACTION=ADD|DELETE|INIT

Specifies the action you want \$QUEUE to perform. The actions are:

##### Action

##### Meaning

##### ADD

Requests \$QUEUE to add an element to the end of the queue.

##### BUILD

Requests \$QUEUE to build a new queue including all the necessary chaining fields.

##### DELETE

Requests \$QUEUE to take an element from the end of the queue and place the address of the element into register 1. If the queue is empty when you specify ACTION=DELETE, register 1 will contain a 0.

**Note:** ACTION= is a required keyword.

**ELEMENT=addr|(Rn)**

Specifies the address, or a register that contains the address, of the element you want to add to the queue.

\$QUEUE uses registers 0, 1, 14, and 15 as work registers. Do not use these registers as values for ELEMENT=. This keyword is valid and required only if you specify ACTION=ADD.

**ERROR=label**

Specifies the label of the routine which receives control if there is an error in the queue. If you do not specify ERROR=, JES2 does not process queue errors.

**HEAD=addr|(Rn)**

Specifies the address, or a register that contains the address, of the queue head.

**Note:** If specified as an address, \$QUEUE places the address in register 2. In this case, do not specify ELEMENT=(R2).

\$QUEUE uses registers 0, 1, 14, and 15 as work registers. Do not use these registers as values for HEAD=.

**Note:** HEAD= is a required keyword.

**SUBPOOL=nnn|251**

Specifies the subpool from which to obtain the storage for the queue elements. This keyword is valid only if you specify ACTION=BUILD.

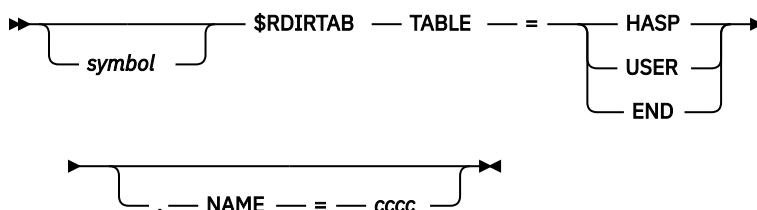
## Environment

- All environments.
- MVS and \$WAITs will not occur.

## \$RDIRTAB – Build table to redirect responses to specific commands

Use \$RDIRTAB to build a table JES2 uses when directing the response to a command to a specific console and/or console area other than the console on which the command was entered.

## Format description



## TABLE=

Specifies the start or end of the redirection table. Use TABLE=HASP to start a redirection table for JES2. Use TABLE=USER to start a redirection table for the user. Use TABLE=END to end a redirection table.

## NAME=

Specifies the 1- to 4-character name of a command group which appears on the REDIRECT(vvvvvvva) initialization statement that defines the console on which the operator enters the command. Valid JES2 command group names (cccc) are:

## DA

## Display active jobs command

## \$REPLYV

### DCON

Display network connections

### DEF

Display JES2 parameter definitions

### DF

Display forms queue

### DI

Display initiators

### DJ

Display job, task (STC), or TSO/E logon (TSU) information

### DN

Display queued jobs

### DNOD

Display NJE nodes

### DQ

Display number of queued jobs

### DSPL

Display spool volumes

### DU

Display units

### LJ

List job output, task (STC), or TSO/E logon (TSU) information

You can also define your own 1- to 4- character command group names in a user redirection table.

The 1- to 4-character command group name in a redirection table should correspond to the name on the CMDRDIR= parameter of the \$SCANTAB macro for the command you are adding to the redirection table.

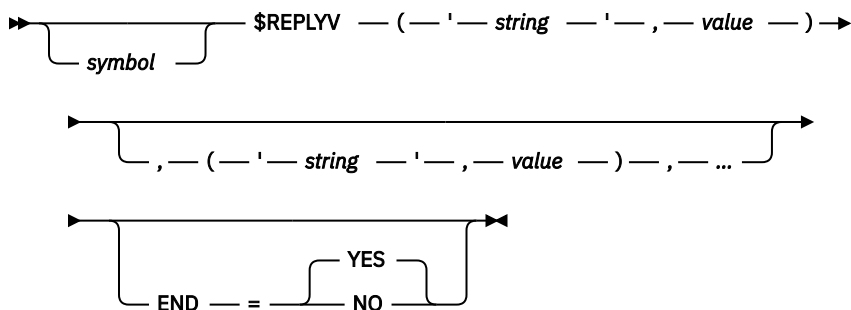
## Environment

- Main task.
- \$WAIT will not occur.

## \$REPLYV – Generate \$REPLYV table entries

Use \$REPLYV to specify valid write to operator (WTOR) replies and process labels. The table generated by \$REPLYV is used by the \$BLDMSG macro to validate the replies to WTORs.

## Format description



### 'string'

Specifies a 1-to 8-character string. This string is compared to the WTOR that is entered by the operator and is used by the \$BLDMSG macro.

This parameter is required and has no default.

**value**

Specifies a value to be used when the string matches the WTOR entered by the operator. This value can include one of the following:

- an address of a routine
- a number (for example, 2 or 4 or 8, to be used as an index)

**END={YES|NO}**

Specifies whether (YES) or not (NO) to generate an end for the table.

## Examples

Following are 2 ways you can code the \$REPLYV macro:

```
$REPLYV (CANCEL,KDRCAN), (CONT,KDRCONT)
```

or

```
$REPLYV (CANCEL,KDRCAN),END=NO
$REPLYV (CONT,KDRCONT),END=YES
```

In both examples, the 2 valid replies are CANCEL and CONT. Control is transferred to label KDRCAN to when 'CANCEL' is replied, and to label KDRCONT when 'CONT' is replied.

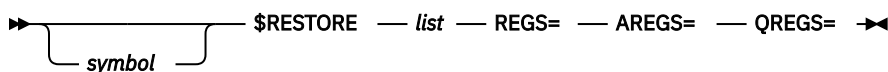
## Environment

- All environments.
- \$WAIT is not applicable - this macro generates a DSECT or a static table entry; it does not generate executable code.

## \$RESTORE – Restore registers from the save area

Use \$RESTORE to restore one or more registers from the current processor's current save area (that is, from the save area built by the most recently issued \$SAVE macro instruction). Or use \$RESTORE to restore one or more registers from the current processor's previous save area.

## Format description

➞  **\$RESTORE** — *list* — REGS= — AREGS= — QREGS= ➞

**list**

Specifies a list of one or more registers, and/or groups of registers to be restored. If more than one register is being restored, the entire list must be enclosed in parentheses.

A register group is indicated by a pair of registers enclosed in parentheses. All registers, beginning with the first register specified and ending with the second register, are restored. The order of restoring a group of registers is: R14, R15, R0-R12. If the list consists of a single group, the outer (list) parentheses are not required.

**Note:** All registers must be specified symbolically. The accepted register symbols are: R0, R1, R2, . . . , R15.

Examples:

```
Restore register 2
$RESTORE (R2) or
$RESTORE R2
Restore registers 15 through 8
$RESTORE ((R15,R8)) or
```

## \$RETABLE

```
$RESTORE (R15,R8)
Restore register 3 and register 10
$RESTORE ((R3),(R10)) or
$RESTORE ((R3),R10) or
$RESTORE (R3,(R10))
Restore registers 0, 3 through 5, and 8
$RESTORE (R8,R0,(R3,R5))
```

**Note:** The sublist order is unimportant.

### REGS=

A list of general purpose registers to be restored.

**Note:** In user and subtask environments, both the general purpose register and the access register are restored.

### AREGS=

A list of access registers to be restored. This parameter is not valid in the USER, FSS, or SUBTASK environment.

### QREGS=

A list of registers for which both the access register and the general purpose register are to be restored. This parameter is not valid in the USER, FSS, or SUBTASK environment.

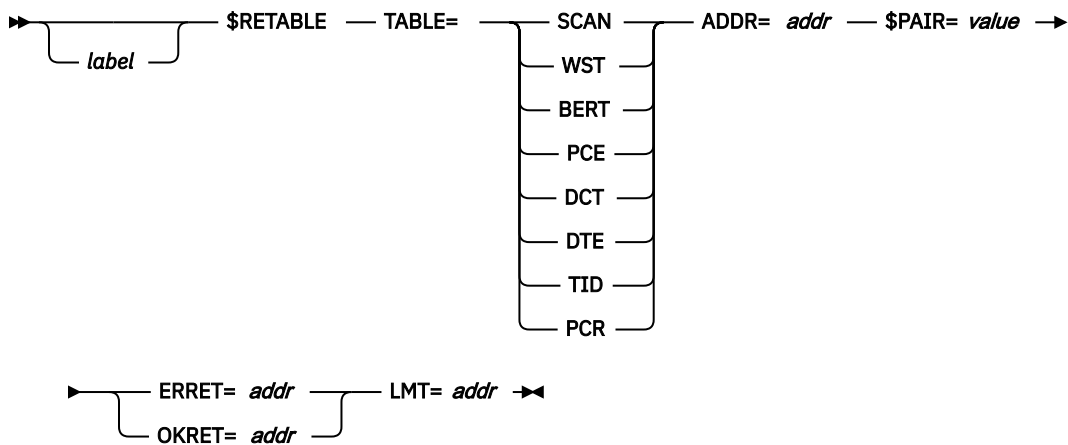
## Environment

- All environments.
- \$WAIT cannot occur.

## \$RETABLE – Remove HASP/user table entry

Use \$RETABLE to remove a table from a MCT table pair. See [Appendix A, “Using JES2 table pairs,”](#) on page 425 for detailed description of table pairs.

## Format description



### TABLE=

Specifies the table pair to be accessed.

```
TABLE=SCAN - $SCANTAB
TABLE=WST - $WSTAB
TABLE=BERT - $BERTTAB
TABLE=PCE - $PCETAB
TABLE=DTE - $DTETAB
TABLE=DCT - $DCTTAB
```

```
TABLE=TID - $TIDTAB
TABLE=PCR - $PCTAB
```

- ADDR=**  
Specifies the address of the table to be removed.
- \$PAIR=**  
Specifies the specific offset and control block of the table pair from which the dynamic table should be removed
- ERRET=**  
The address to go to if the table could not be removed.
- OKRET=**  
The address to go to if the table is successfully removed.
- LMT=**  
Specifies the address of the LMT of the module whose tables are removed.

Return codes

The following return codes (in decimal) are returned in register 15:

Return Code	Meaning
0	Requested table entry was found.
4	Requested table entry was not found or end-of-table.

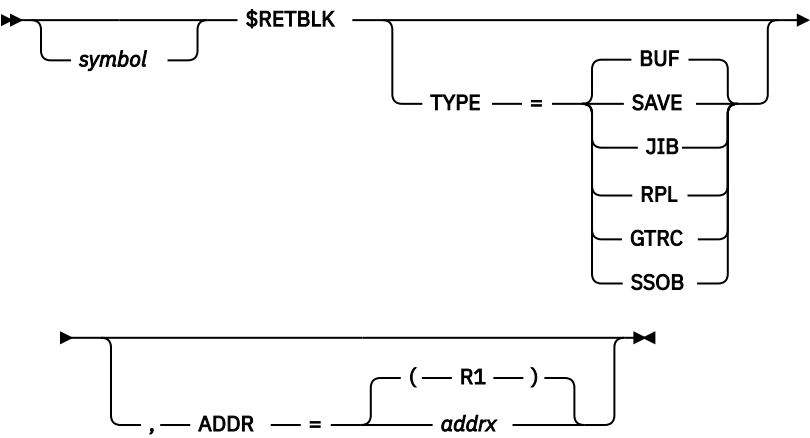
Environment

- Main task

\$RETBK – Return a storage cell to a free–cell pool

Use \$RETBK to return a number of predefined storage cells to one of several previously established free–cell pools.

Format description



- TYPE=**  
Specifies the type of storage cell to be returned to the free cell pool. Specifications are as follows:
- | Specification | Meaning |
|---------------|---------|
|---------------|---------|

## \$RETSAVE

### SAVE

An MVS-type save area

### JIB

A JOE information block

### BUF

A 4K I/O buffer

### RPL

A request parameter list control block chain

### GTRC

A GETREC chain control block

### SSOB

A subsystem options block

### ADDR=

Specifies the address of the first cell to be returned. If register 1 is used, it must contain the address of the first cell to be returned before executing this macro.

## Environment

- Functional subsystem (HASPFSM).
- MVS WAIT cannot occur.

## \$RETSAVE – Return a JES2 save area

---

Use \$RETSAVE to return the current JES2 save area(s) to the JES2 free pool of save areas.

## Format description



### FRETRE=

Specifies the label or a register that contains the address of the task control block (TCB) recovery element (TRE) that is associated with the save area(s) that are being returned to the JES2 free pool. FRETRE= is a valid specification only in the user environment.

## Environment

- JES2 main task.
- \$WAIT cannot occur.

## \$RETURN – Restore registers, free the JES2 save area, and return to the caller

---

Use \$RETURN to restore the caller's registers saved in the current processor save area and return that save area to the save area pool.

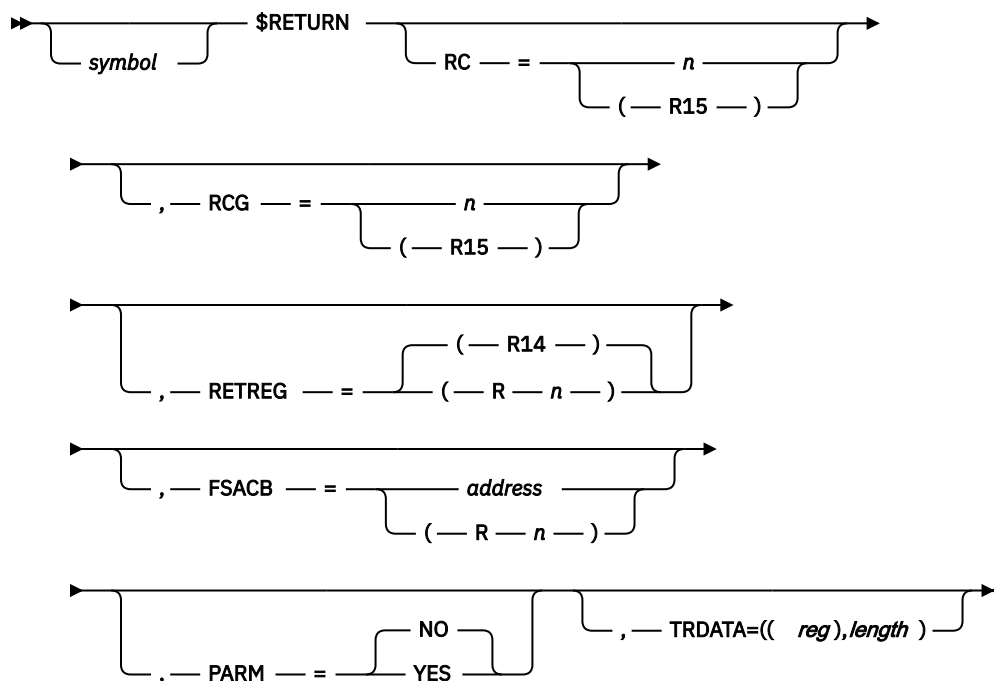
### Note:

1. If this macro is coded within a functional subsystem environment, \$RETURN generates inline code to place the save area pointed to by register 13 onto the unused save area stack. All necessary code for standard linkage conventions is also generated.
2. Tracing for \$RETURN is controlled through the TRACE= parameter on the \$SAVE macro.



3. In the JES2 and USER environments, the ASC mode at the time the \$SAVE is also restored.
4. Access registers AR0, AR1, and AR15 are not affected by the \$RETURN macro. In the USER environment, all other access registers are restored. In the JES2 environment, access registers are restored if the caller was in AR ASC mode at the time of the \$SAVE.
5. In the FSS environment, AR ASC mode callers are not supported.

## Format description



### RC=

Specifies a numeric return code to be returned in register 15. If this operand is not specified, the return code is set to 0. RC=0 is the default.

If register 15 is used, the return code value must be loaded into register 15 before the execution of this macro instruction.

### RCG=

Specifies a numeric 64-bit return code to be returned in register 15. If this operand is not specified, the return code is set to 0. RC=0 is the default. If register 15 is used, the return code value must be loaded into register 15 before the execution of this macro instruction.

### RETREG=

Specifies a register used to save the return address. R14 is the default.

RETREG= is valid only in the JES2 main task environment.

### FSACB=

Used to determine whether processor tracing is on. The use of this keyword is only valid in the functional subsystem environment. If an FSACB specification is not provided, only global \$RETURN tracing is done.

### PARM=

Specifies whether (YES) or not (NO) \$RETURN processing should skip over an inline parameter list when control is returned to the calling module from the current save area. Return is through register 14.

**Note:** The first byte of the parameter list must contain the length of the parameter list. (The length must also include this first byte.)

## \$RETNWORK

### TRDATA=

Additional data to be included in \$RETURN trace entry. Only valid if TRACE=YES. TRDATA=((*reg*),*len*) is the format. Where *reg* is register that points to data to trace. *len* is either a register (that is, (*len*)) or a length (that is, *value*) or an offset in the area pointed to by *reg* (that is, +*offset*). In either case *reg* can be Rx (low half of a register), Hx (high half of a register), or ARx (access register). The max amount of data that is traced is 128 bytes. Register values used for TRDATA are those that were saved by \$SAVE unless a \$STORE was done.

### Note:

1. If this macro is issued from the main task environment when a PRE (\$ESTAE) currently exists for the save area level about to be returned, the \$ESTAE is canceled.
2. The use of the \$RETURN macro assumes that register 11 contains either the address of the HASP communication table (HCT) for the JES2 main task and subtask environment, the HASP function communication table (HFCT) for a functional subsystem module or the HASP common communication table (HCCT) for a user task environment.

## Environment

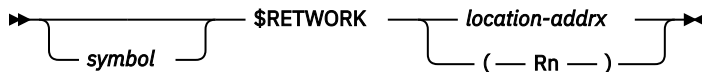
- All environments.
- \$WAIT cannot occur.
- Callers in AR ASC mode are supported in the JES2 and USER environments.

## \$RETNWORK – Return a work area

---

Use \$RETNWORK to return a work area obtained with the \$GETWORK macro instruction.

### Format description



### location

Specifies the address of the work area to be returned. (This address is loaded into register 1.)

The work area to be returned must have been obtained through a previous \$GETWORK macro instruction.

## Environment

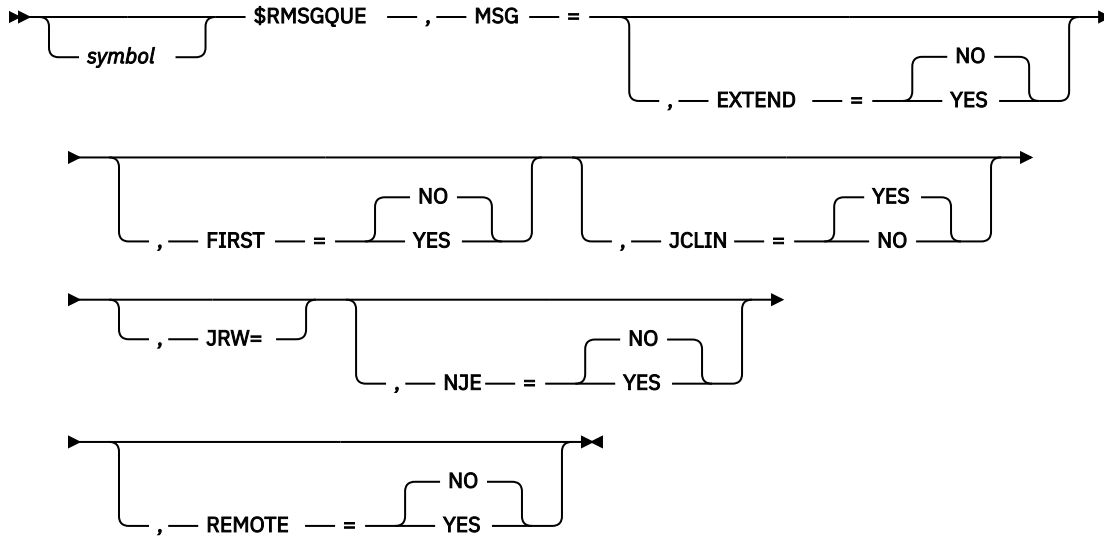
- Main task.
- \$WAIT cannot occur.

## \$RMSGQUE – Queue message to JES2

---

Use \$RMSGQUE to interface with the RMSGQUE service that queues a message RJCB for processing.

## Format description



### MSG=(*addr,type*)

Specifies the message.

Where:

#### *addr*

Is the address of the message to issue. The format of the message is:

```
DC      Y(Length of following message)
$MSG nnn,'text'
```

*text* is assumed to begin with "--" if JCLIN=YES is specified.

#### *type*

Is optional. *type* can be either ERROR, WARNING or DEFER. If not specified and JCLIN=YES, then ERROR is assumed.

ERROR implies the job fails conversion. WARNING means this is merely a warning message.

DEFER means that the message is added to JCLIN, but no WTO is issued until the job is completely read. Use DEFER if the error is node-dependent. If the job is later determined to be destined for a different node, the deferred message is deleted. DEFER implies ERROR.

**Note:** If JCLIN=YES is specified, the message text must begin with two minus signs (--).

### EXTEND=

Specifies whether (YES) or not (NO) the message is extended with source information if it is issued to the console. The default is EXTEND=NO.

### FIRST=

Specifies the position where the message is added. The default is FIRST=NO.

#### **YES**

The message is added to the front of the message queue.

#### **NO**

The message is added at the end of the message queue.

### JCLIN=

Specifies whether (YES) or not (NO) the message is written to the JCLIN data set. The default is JCLIN=YES.

**Note:** If JCLIN=YES is specified, the message text must begin with two minus signs (--).

**\$RUSE**

**JRW=**

Specifies the address of the JRW. If the JRW is not specified, it is assumed to be addressable.

**NJE=**

Specifies whether (YES) or not (NO) the message is sent to the NJE origin. The default is NJE=NO.

**Note:** If NJE=YES is specified, the input must be an NJE device (NJHGORG, NJHGORGQ, NJHGORGGR).

**REMOTE=**

Specifies whether (YES) or not (NO) the message is sent to a remote. The default is REMOTE=NO.

**Note:** If REMOTE=YES is specified, the input must be an RJE device.

**Register usage**

The following table shows the register usage of \$RMSGQUE macro:

REG	VALUE ON ENTRY TO MACRO	VALUE ON EXIT
R0	n/a	destroyed
R1	n/a	queued RJC address
R2-R10	n/a	unchanged
R11	HCT/HCCT address	unchanged
R12	n/a	unchanged
R13	SAVE area	unchanged
R14-R15	n/a	destroyed

**Return codes (R15 on exit)**

None.

**Environment**

Main task.

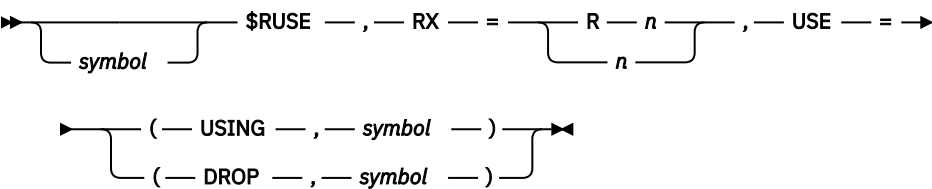
**\$RUSE – Establish USING on a register**

Use \$RUSE to establish or drop a USING on a register and document that register usage. If you cannot use an actual USING on a register and the routine relies on the register's contents for a lengthy duration, use \$RUSE. JES2 establishes a *dummy* DSECT with the label \$RUSE in \$MODULE. The macro equates the symbol you supply to the DSECT and then generates a USING.

**Note:**

1. You must code \$MODULE in any module issuing \$RUSE.
2. You can only specify one \$RUSE per register to be in effect at any one time.
3. You cannot issue \$RUSE for a symbol that is already defined in a module except for a symbol defined by a prior \$RUSE invocation.

**Format description**



**RX=**

Specifies the register (0-15) for which JES2 establishes or drops the USING.

**USE=**

Specifies whether this macro invocation is to establish a register USING or DROP a previous register USING.

**USING,symbol**

Indicates that this macro call is to establish a register USING. If the specified register is currently *in use* due to a prior \$RUSE call, JES2 drops the register and then establishes this USING.

*symbol* is required when you specify USE=USING.

**DROP[,symbol]**

Indicates that this macro call is to drop a previously established USING. If \$RUSE was not used to establish the USING for the specified register, JES2 does not drop the USING.

*symbol* is optional when you specify USE=DROP. However, if you specify a symbol, JES2 confirms that the previous USE=USING for this register matches this symbol. If the symbols do not match, JES2 issues an MNOTE that indicates there was no previous USING for this symbol.

## Environment

- All environments.
- \$WAIT cannot occur.

## \$SAVE – Obtain JES2 save area and save registers

---

The \$SAVE macro instruction obtains a register save area from a JES2-managed save area pool and saves registers 14, 15, and 0 through 12 in the save area. No registers are destroyed by executing this macro. If this macro is coded within any environment except the JES2 main task, \$SAVE will save the registers in the save area pointed to by register 13. All necessary code for standard linkage conventions is also generated. For information on linkage conventions, see [z/OS JES2 Installation Exits](#).

Access registers are always saved in the USER environment. In the JES2 environment, if the callers of \$SAVE is in AR ASC mode, then access registers are saved. The ASC mode upon return from the \$SAVE macro in the JES2 and USER environment depends on the assembly time global symbol &SYSASCE (set by the MVS SYSSTATE macro) as follows:

**If SYSASCE=**

**then \$SAVE returns in**

**P**

primary ASC mode

**AR**

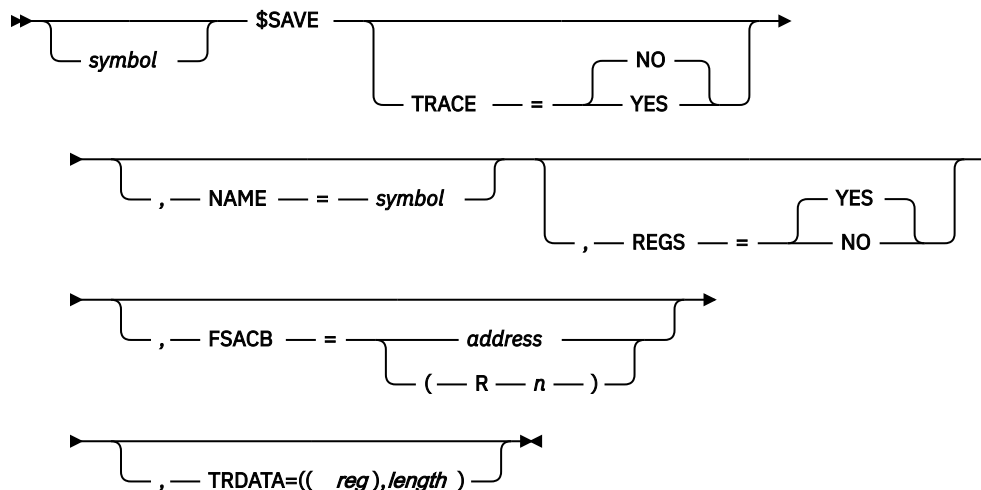
AR ASC mode

**ANY**

the ASC mode of the caller of \$SAVE

In the FSS environment, AR ASC mode callers are not supported.

## Format description



### **TRACE=**

Specifies whether the \$SAVE macro and corresponding \$RETURN macro are traced as follows:

#### **YES**

##### **For \$SAVE:**

Specifies the PCE address; the contents of registers 14, 15, 0 and 1; and an 8-character symbol designating where the \$SAVE macro instruction was issued and traced.

##### **For \$RETURN:**

Specifies that \$RETURN is traced through the JES2 event trace facility. The PCE address and the returned registers, 14, 15, 0, and 1, are traced (TRACE ID=2 in the JES2 main environment or TRACE ID=19 in the user environment), and an 8-character symbol designating where the \$RETURN macro was issued is also traced.

#### **NO**

Indicates that the \$SAVE and \$RETURN macro instructions are not traced. This is the default.

### **NAME=symbol**

Specifies the name associated with this \$SAVE macro for tracing and diagnostic purposes. If NAME is not specified, the label (symbol) on the \$SAVE macro is used for identification. If neither NAME or symbol is specified, the current CSECT name is used.

### **REGS=**

Specifies whether a starting store multiple instruction (STM) and ending load multiple instruction (LM) are used. This specification is not valid in JES2 main task.

### **FSACB=**

Used to determine whether processor tracing is on. The use of this keyword is only valid in the functional subsystem environment. If an FSACB specification is not provided, only global \$RETURN tracing is done.

### **TRDATA=**

Additional data to be included in \$SAVE trace entry. Only valid if TRACE=YES. TRDATA=((*reg*),*len*) is the format. Where *reg* is register that points to data to trace. *Len* is either a register (that is, (*len*)) or a length (that is, *value*) or an offset in the area pointed to by *reg* (that is, *+offset*). In either case *reg* can be Rx (low half of a register), Hx (high half of a register), or ARx (access register). The max amount of data that is traced is 128 bytes. TRDATA= cannot be specified if REGS=NO.

**Note:** The use of the \$SAVE macro assumes register 11 contains either the address of the HASP communication table (HCT) for the JES2 main task and subtask environment, the HASP function communication table (HFCT) for a functional subsystem environment, or the HASP common communication table (HCCT) for a user task environment.

**Attention:**

The TRACE=YES parameter is provided so that normal JES2 operations are traced through the JES2 trace facility. Most JES2 central services specify TRACE=YES on their respective \$SAVE macros. Individual routines issuing the \$SAVE macro should not specify TRACE=YES if the routine is called an excessive number of times, unless the routine is considered an part of the normal JES2 logic flow. If a routine is traced on an infrequent basis, a trace ID can be assigned to that function so it can be traced independently.

Also, you must have stored in register 11 the address of the HCT (or the HFCT if running in an FSS environment) before executing this macro.

## Environment

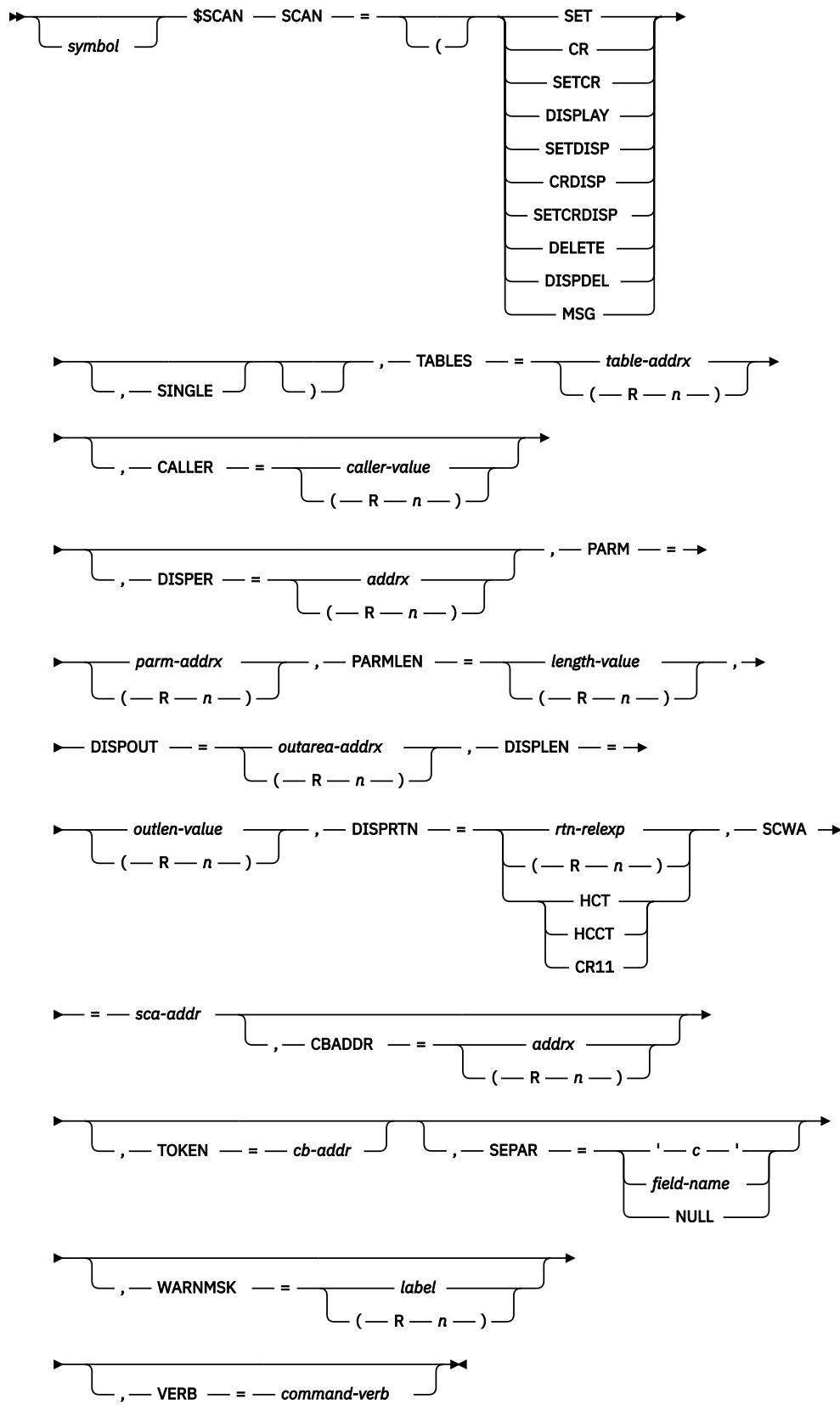
- All environments.
- AR mode callers are not supported in the HASPFSSM address space.
- \$WAIT cannot occur in the main task.
- MVS WAIT can occur in the user, subtask, and FSS environments.

## **\$SCAN – Scan JES2 parameter statements**

---

Use \$SCAN to scan JES2 parameter statements and set, or display the appropriate fields, as defined by the scan tables (\$SCANTAB) passed.

## Format description



### SCAN=

Indicates the scan type. There are two positional parameters. The first can have the following values:



**SET**

A SETTING scan is requested to scan the input and SET control block fields. For SCAN=SET, The DISPLEN and DISPRTN operands are not required but should be used to allow display of possible diagnostics.

New elements cannot be created by SET, for example, 'FSSDEF(XYZ)' to reset some of the attributes of the FSSDEF named XYZ would be allowed, but if XYZ did not exist, the call would return an error.

**CR**

A CREATE SETTING scan is requested, which is similar to the SET call except new elements must be created. For example, 'FSSDEF(XYZ)' to create the new FSSDEF called XYZ would be allowed, but if XYZ already existed, the call would return an error.

**SETCR**

This is a SETTING call that allows resetting or creating, for example, either the SET or CR case.

**DISPLAY**

A DISPLAY scan is requested to scan the input and display the CB fields that were implied. For SCAN=DISPLAY, the DISPLEN and DISPRTN operands are required. For example, \$D commands use DISPLAY.

**SETDISP**

A SETTING scan operation is to be done immediately followed by a DISPLAY operation to show the results of the SET all within one scan call. This requires the DISPLEN and DISPRTN operands. For example, \$T commands use SETDISP.

**CRDISP**

A CREATE SETTING scan operation is to be done immediately followed by a display operation to show the results of the set, all within one scan call. This requires the DISPLEN= and DISPRTN= operands. For example, \$ADD commands use CRDISP.

**SETCRDISP**

A SETCR SETTING scan operation is to be done immediately followed by a display operation to show the results of the set, all within one scan call. This requires the DISPLEN= and DISPRTN= operands.

**DELETE**

A DELETE operation is to be performed similar to a SET operation, except that the element is to be deleted rather than modified.

**DISPDEL**

A DISPLAY operation followed by a DELETE operation. The display is done before the delete in this case because there would be nothing to display after the delete.

**MSG**

Indicates that a MESSAGE BUILDING scan is requested. This is usually done as a result of the \$BLDMSG macro. This requires the DISPLEN= and DISPRTN= operands.

**SCAN= (2)**

The optional second operand of SCAN= can specify 'SINGLE'. Use this keyword to limit the scan to a single high-level keyword.

**TABLES=**

Specifies the address (label or R2-R12) of a \$PAIR containing the addresses of the scan tables (\$SCANTAB) that will be used in order when scanning the statement. Either one or the other word can be zero but not both.

**CALLER=**

Specifies an optional caller ID (value or R2-R12) for use during scanning. If the optional caller ID is specified, only the scan table (\$scantab) entries that specify that caller ID in their CALLER= operands and the entries that do not specify CALLER= is used for this \$SCAN call.

A second positional value can be specified for this operand. It specifies the caller ID to use for the display request that follows the setting operation, when SCAN=SETDISP, SETCRDISP, and so on. If

## **\$SCAN**

this is not specified, the CALLER= value is used as the default for the display. Then, register values for the caller IDs require extra parentheses, for example, CALLER=((RX),XYZ).

Internally \$SCAN uses the second value during the display request within SETDISP, and so on; therefore, SCWACALR/SCWACALD are the set and display caller IDs, respectively, during set. But then during the subsequent display scwacalr/scwacald are the display caller ID and the original set caller ID, respectively.

### **DISPER=**

Specifies an optional display ID (addr or R2-R12) for use during scanning. If the optional display ID is specified, only the scan table (\$SCANTAB) entries which specify a display id in their DISPERS= operands which correspond to that specified by this \$SCAN call and the entries that do not specify DISPERS= will be used for this \$SCAN call. A correspondence between a \$SCANTAB entry and a \$SCAN caller exists if each bit in one of the \$SCANTAB DISPER entries is on in the mask provided by the caller.

### **PARM=**

Specifies the address (label or R2-R12) of the parameter statement to be scanned.

### **PARMLEN=**

Specifies the length (value or R2-R12) of the parameter statement pointed to by PARM.

### **DISPOUT=**

Specifies the address (label or R2-R12) of the output area to be used for display requests. If this is not specified and DISPLEN is specified, an output area is dynamically allocated.

### **DISPLEN=**

Specifies the length (value or R2-R12) of the output area pointed to by DISPOUT; or, if DISPOUT is not specified, it indicates the length of storage \$SCAN should allocate for a display output area to be used in the \$SCAN call.

DISPLEN is required for all call types that imply displays, and is suggested even for non-display calls so that because of an error, the diagnosis can be displayed.

### **DISPRTN=**

Specifies the address (label or r2-r12) of the routine to call to display the output. R1 contains the address of the scan work area (SCWA) on entry to this routine. The SCWA points to the output area and so on.

DISPRTN is required for all call types that imply displays, and is suggested even for non-display calls so that diagnosis can be displayed in the case of errors.

You can use a second positional parameter to define R11 on entry to the routine.

### **HCT**

HCT address.

### **HCCT**

HCCT address.

### **CR11**

R11 at time of call (or the value specified on CR11= parameter).

If the value of the second positional is not specified, the default value is as follows:

### **HCT**

Indicates that the routine is found in the PADDR.

### **HCCT**

Indicates that the routine is found in the CADDR.

### **CR11**

Indicates that the routine is found in all other cases.

### **CBADDR=**

Specifies the oldest parent control block. If the older parent control block is specified, \$SCAN supplies this address for a \$SCANTAB at the highest table level if that \$SCANTAB specified CB=PARENT.

**TOKEN=**

Used to pass a value (usually an address) to scan exits such as PRESCAN, POSTSCAN, and DISPLAY.

**SEPAR=**

Used to specify what separator character should be used between fields during display processing. The specifications are:

1. One character enclosed within apostrophes.
2. The word 'NULL' (not within apostrophes).
3. Any field name suitable for an RR instruction.

If form (1) is used, that is the character used. If form (2) is used, there is no separator. If form (3) is used, the separator character is the same as the contents of the specified field.

This is an optional keyword.

**WARNMSK=**

Specifies a byte mask, probably using \$SCWXXXX equates, that is compared against the WARNING= in the \$SCANTABS scanned. If any bits match, the parameter statement is ignored, that is, not scanned, and a return code 4 (warning) will be returned from the \$SCAN call.

**Note:** The first 6 bits of this mask are reserved for IBM indications, and the rightmost 2 bits for installation indications. The first bit, X'80', is always turned on by \$SCAN to indicate that obsolete parameters result in a warning. The remaining bits are defined in \$HASPEQUS.

**VERB=**

Points to a command verb to be used in keyword tracebacks on the \$HASP003 message in case of an error. The verb passed to \$SCAN consists of 1 1-byte length field followed by the verb. If VERB= is not specified, no verb is added to the \$HASP003 keyword trace.

**Note:** The \$SCAN macro issues a \$GETWORK macro internally to get a dynamic work area for its parameters, and so on. The \$SCAN routine issues the corresponding \$REWORK.

## Environment

- JES2 main task only.

## Registers on entry

**R0 - R10:**

N/A

**R11:**

HCT base address.

**R12:**

N/A

**R13**

PCE base address.

**R14 - R15:**

N/A

## Registers on exit

**R0 - R1:**

Destroyed

**R2 - R13:**

Unchanged

**R14**

Destroyed

**R14**

Destroyed

**R15:**

Return code

**Return codes**

The following return codes (in decimal) are returned in register 15:

**Return Code****Meaning****0**

Scan successful.

**4**

Parameter statement obsolete.

**8**

Parameter statement not supported in tables.

**12**

Parameter statement contains errors.

The following return codes (decimal) are for PRESCAN:

**Return Code****Meaning****4**

TERMINATE SCAN, RESTORE

**8**

DONE, GET NEXT INPUT

**12**

SKIP THIS ENTRY IN LOOP

**16**

FILTER NOT EQUAL

**20**

FILTER GREATER

**24**

FILTER LESS

**28**

FILTER NEVER MATCHES

**Other considerations**

- \$WAIT and WAIT cannot occur unless SCAN=MSG.
- \$WAIT or WAIT can occur if SCAN=MSG depending upon options are passed to the DISPRTN through the \$BLDMSG macro.
- Every command passed to \$SCAN must have a trailing blank space.

**\$SCANB – Backup storage for a scan**

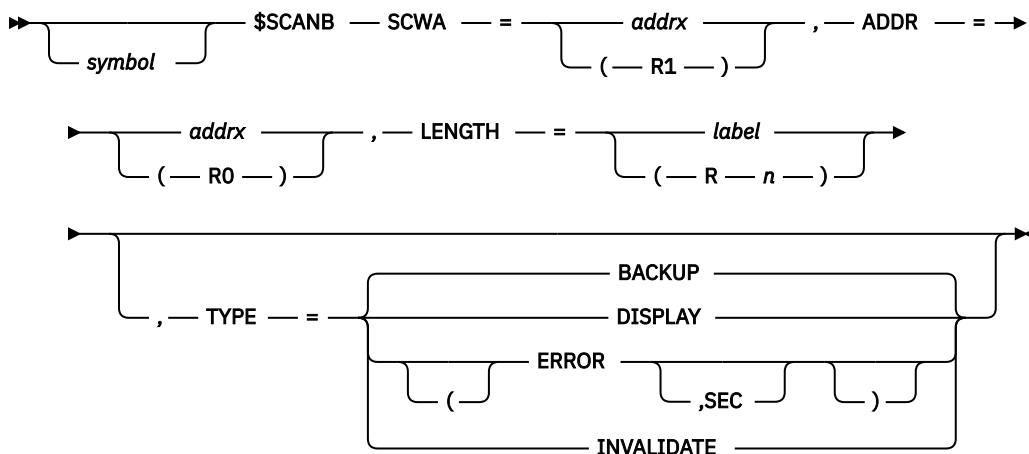
---

Use \$SCANB to backup a copy of a storage area before it is possibly changed during execution of the \$SCAN facility. \$SCANB may be used only within a prescan or postscan exit routine specified through the PRESCAN and PSTSCAN operands of the \$SCANTAB macro.

The \$SCAN facility uses \$SCANB to backup all control block fields before they are changed. If, at any time during the scan, an error is found, \$SCAN uses the backups created by \$SCANB to restore all the changed

fields to their contents before the start of the scan. If a \$SCAN prescan or postscan exit routine changes a storage area, it should first backup that area using the \$SCANB macro.

## Format description



### SCWA=

Specifies the address of the current scan work area, mapped by the \$SCANWA macro.

### ADDR=

Specifies the address of the storage area to backup before the scan possibly changes it.

### LENGTH=

Specifies the length (in bytes) of the storage area indicated by the ADDR operand. If register notation is used, only registers 2 through 12 are valid.

### TYPE=

Specifies that an area of storage is to be used following a SET and DISPLAY \$SCAN request or if an error occurs within a \$SCAN call.

#### DISPLAY

Indicates to save an area of storage to use to display the results of a SCAN=SET request. The value that is set is passed to \$SCAN and used as input for a SCAN=DISPLAY request, for example:

```
$SCANB SCAN=DISPLAY, ADDR=addrx, LENGTH=
```

This value must, therefore, also be scannable by \$SCAN.

#### ERROR

Indicates to save an area of storage to use if an error is encountered during a \$SCAN call. \$SCAN then returns this keyword value to point to the location of the error.

Optionally, a second operand can be supplied with ERROR:

#### (ERROR,SEC)

Indicates that this is a secondary error string.

#### BACKUP

Indicates to produce a backup copy of the storage area before it is possibly changed during the execution of \$SCAN.

#### INVALIDATE

Indicates that any TYPE=BACKUP areas that have been created for the specified range of storage are no longer valid.

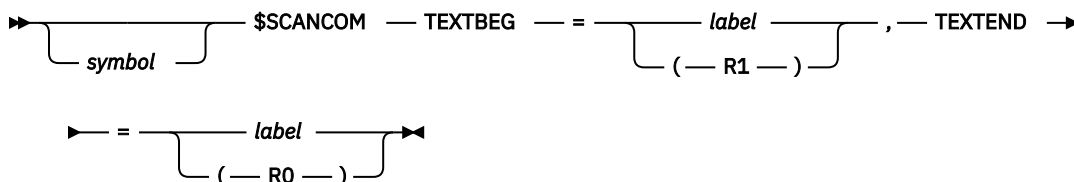
## Environment

- Not applicable.

## \$SCANCOM – Call the \$SCAN facility comment service routine

Use the \$SCANCOM macro to search for and locate the first non-blank, non-comment character in a specified text string. This facility allows the \$SCAN facility to ignore (skip over) comment text provided in both initialization statements and commands. A return code is passed in register 15. JES2 returns the address of the non-blank, non-comment character in register 1.

## Format description



**TEXTBEG=**

Specifies the address of the beginning of the text that is to be scanned by the \$SCAN facility.

**TEXTEND=**

Specifies the address of the end of the text that is to be scanned by the \$SCAN facility.

## Return codes

The following return codes (in decimal) are returned in register 15.

### Return Code Meaning

**O**

Non-blank and no comments found

4

Valid comment, non-blank found

8

End of statement encountered, no non-blanks, or non-comment characters found

12

No asterisk-slash (\*/), the comment ending delimiter, found and an invalid comment encountered

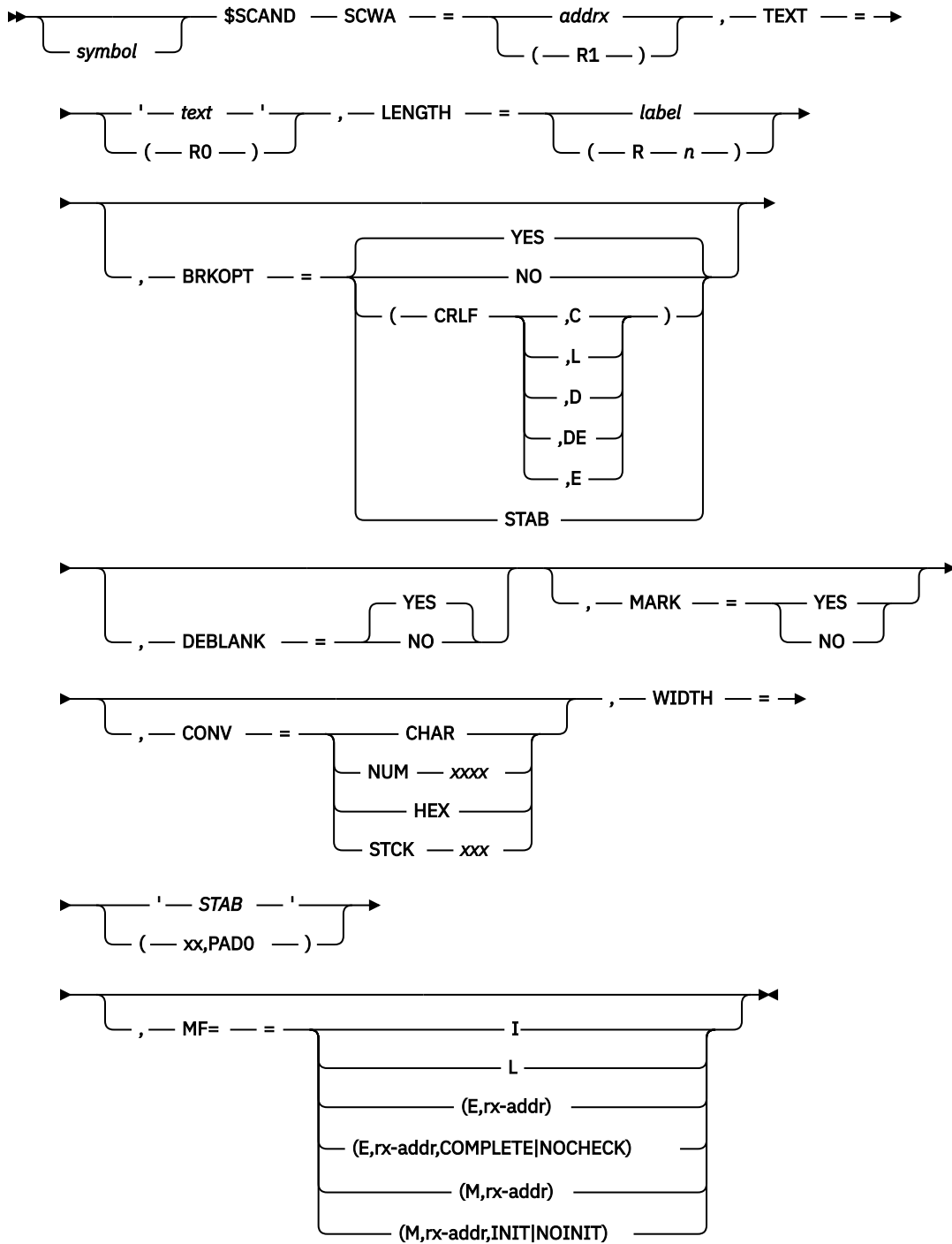
## Environment

- JES2 main task.
- \$WAIT cannot occur.
- Every command passed to \$SCANCOM must have a trailing blank space.

## \$SCAND – Call the \$SCAN facility display service routine

Use the \$SCAND macro instruction to call the display service exit routines that are called by \$SCAN to add text to a display line being created for the SCAN=DISPLAY request. This macro instruction can only be called from a \$SCAN exit routine or \$SCAN itself.

## Format description



### SCWA=

Specifies the address of the current scan work area. This can either be provided as the actual address, a label, or a register (R1-R12).

### TEXT=

Specifies the text (specified in single quotation marks) to be added to the display line or the address of that text as specified in a register (R2-R12).

**LENGTH=**

Specifies the length of the text to be added. This can either be a label or a register (R2-R12). If the actual text is provided on the TEXT= keyword, the length specification on this keyword defaults to the length of that text.

**BRKOPT=**

Specifies that the text that is specified by the TEXT= keyword is separated or not from the text already passed.

**YES**

Indicates that the text that is specified by the TEXT= keyword is separated from the text already passed.

**NO**

Indicates that the text that is specified by the TEXT= keyword is not separated from the text already passed.

**CRLF**

Indicates same as BRKOPT=YES, but also requests that this display start on a new line. A second optional value is the line type for multi-line WTOs. Valid values are C, L, D, DE, and E. These are only valid if BRKOPT=CRLF. See the WTO macro for complete descriptions.

**STAB**

Indicates that the \$SCANTAB settings are to be used to determine whether a new line is needed. Also, indicates that line type settings are to be determined from the \$SCANTAB.

**DEBLANK=**

Specifies whether (YES) or not (NO) the blanks and X'00's are to be removed from the front and end of the text.

**MARK=**

Specifies whether (YES) or not (NO) the location of the text should be remembered in case the display at this \$SCAN level must be backed out because a later display filter did not match. If you specify MARK=YES, you must also have specified BRKOPT=YES.

**CONV=**

Specifies conversion.

**CHAR=**

Specifies a character string of the text passed to \$SCAND. This string does not require conversion except for possible trimming of blanks.

**NUMxxxx**

Indicates that the keyword represents a numeric value that is stored in binary. The optional second positional defines a multiple to round the input value to before storing. The optional third positional defines a value by which the value of the keyword is multiplied before storing. For example, CONV=(NUM,8,100)).

The xxxx can be specified to the following values:

**T**

Indicates a thousands separator (,) is used to display the numeric value. For example, 1,234 instead of 1234.

**U**

Indicates that the value in the field is treated as an unsigned integer.

**S**

Indicates that the value in the field is treated as a signed integer.

**\***

Indicates that an asterisk (\*) is displayed when the value in the fullword numeric field is X'FFFFFFFF'.

**HEX=**

Specifies a hexadecimal number that was passed in the text string. This number must be converted to printable.



**STCKxxx=**

Specifies a time stamp in STCK or STCKE format that must be displayed in the format *yyyy.ddd, hh:mm:ss*.

The xxx indicates additional options regarding the format of the field and the display. It can be specified the following values:

**E**

Indicates that the field is a STCKE, not a STCK.

**U**

Default display but without parenthesis.

**L**

Indicates a Long display in the format day mon year AT hh:mm:ss.

**W**

Includes day of week (Long display only) dayname day mon year AT hh:mm:ss.

**M**

Displays time to minute precision only.

**Z**

Indicates that the STCK value must be converted to local time.

**0,1,2,3,4,5,6**

Indicates the number of decimal places of precision the second must be displayed to. This specification is not honored for 4-byte fields.

**A**

Indicates that the actual value to be displayed or filtered is not the time stamp itself but the difference (in seconds) between the STCK value and the current time. The same as CONV=NUM, the second and third positionals are scaling and rounding values. For example, CONV=(STCKA,,60) gives minutes.

**T**

Indicates that the age must be displayed with a thousands separator (,). It is used together with A.

**WIDTH=**

Specifies the number of places in the output area (from 1 to 32) to be used to display the results. It is only valid with CONV= NUM (right-aligned), HEX (left-aligned), CHAR (right- or left-aligned), or STCK (left-aligned unless STCKA that is right-aligned). A value of zero indicates that no specific width is associated with the display.

A second operand on width, PAD0, (valid only for CONV=NUM) indicates to pad the number with leading 0s.

**MF=**

Specifies the macro form. It can be specified to the following values:

**MF=I**

Specifies the inline form.

**MF=L**

Specifies the list form.

**MF=(E,rx-addr) or MF=(E,rx-addr,COMPLETE|NOCHECK)**

Specifies the execute form:

- The rx-addr is the address of an MF=L form. The address can be specified using register notation (R1 through R12) or as a rx-addr expression.
- COMPLETE (default) specifies that \$BLDMSG is to check for required parameters and supply optional parameters that you did not specify.
- NOCHECK specifies that \$BLDMSG does not check for required parameters and does not supply the optional parameters that you did not specify.

## \$SCANDIA

### MF=(M,rx-addr) or MF=(M,rx-addr,INIT|NOINIT)

Specifies the modify form:

- The rx-addr is the address of an MF=L form. The address can be specified using register notation (R1 through R12) or as a rx-addr expression.
- INIT|NOINIT specifies whether to initialize (INIT) or not (NOINIT) the \$BLDMSG parameter list to the MF=L defaults before setting parameter values. The default is NOINIT.

## Environment

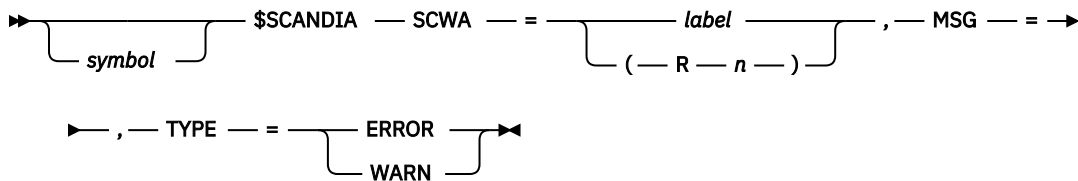
- JES2, USER, SUBTASK, or FSS (HASCSCAN environment only) main task.
- \$WAIT cannot occur.

## \$SCANDIA – \$SCAN diagnostic message service

---

Use \$SCANDIA to issue a diagnostic message during scan processing (that is, in the HASCSCAN environment). You can call this service both prescan and postscan exits to issue warning- or error-level messages.

## Format description



### SCWA=

Specifies the address as a label or a register (R1-R12) that contains the address of the SCWA (scan work area) to be used for the scan level.

### MSG=

The diagnostic message skeleton of the message text, mapped as follows:

#### Offset

##### Use

#### +0

2-byte reason code (EBCDIC or numeric)

#### +2

1-byte text length

#### +3

text

### TYPE=

Specify the message type as one of the following:

#### WARN

Indicates that JES2 will issue a warning message. \$SCAN processing continues.

#### ERROR

Indicates that JES2 will issue an error message. \$SCAN processing ends.

## Environment

- JES2, USER, SUBTASK, or FSS main task (HASCSCAN environment only).
- \$WAIT can occur.

## \$SCANTAB – Create a scan table

Use \$SCANTAB to create scan tables to be used with the \$SCAN facility, defining the allowed input and syntax for initialization parameter statements, selected messages, and some operator commands. JES2 uses \$SCANTAB to define the initialization parameter statements, initialization options, selected messages, and selected operator commands.

\$SCANTAB entries are used to define the start of a user table (\$SCANTAB TABLE=USER...) or a JES2 table (\$SCANTAB TABLE=HASP...), the end of a table (\$SCANTAB TABLE=END) or an entry in a table (\$PCETAB NAME=JJ2...). Each entry defines:

- An operand allowed in the statement input. The operand can be either a keyword operand (for example, OUTDisp= on the OUTCLASS(v) initialization statement) or a coded-value operands (for example, BURST on the PRT(nnnn) initialization statement). If you need to display the value of an operand, use keyword operands because \$SCAN only displays this type of operand.
- How to find the correct control block and fields related to the operand.
- What the allowed input can be.
- How to convert the input for storing into the fields or convert the contents of the fields for display. Because \$SCANTAB generates only tables, not executable code, register notation may not be used for any of the operands.

By default, the \$SCANTAB macro does not expand the table entry in the assembly listing. If you require this information use either of the following methods:

- Assemble your module with:

```
$MODULE SYSP=(PRINT,GEN,DATA,NOGEN,NOGEN)
```

- Set the SYSPARM keyword on the EXEC statement as:

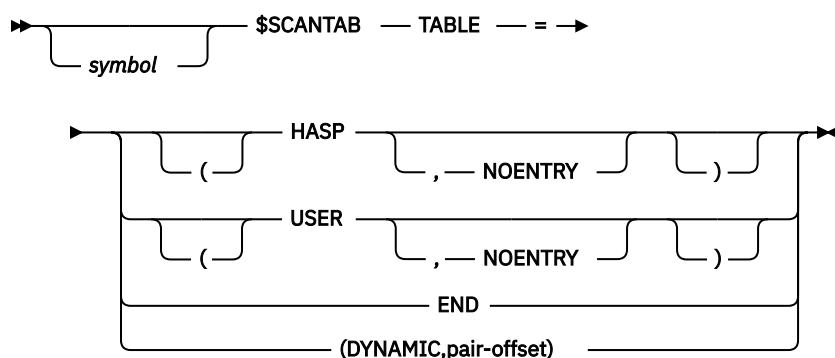
```
EXEC ASMA90,PARM='SYSPARM=(PRINT,,,)'
```

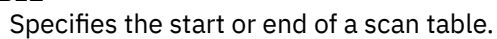
**Note:** The format description that follows breaks the macro into a **boundary form** (the form that starts or ends a table) and an **entry form** (the form that defines each table entry).

## Format description

The following format descriptions apply:

### Boundary form



**TABLE=**

Specify TABLE=HASP or TABLE=USER to start the corresponding table. Optionally specify a second parameter of NOENTRY, for example, TABLE=(USER,NOENTRY)), to indicate that no ENTRY statement need be generated for the label of the scan table.

DYNAMIC specifies that this is a dynamic table. The second and subsequent positionals, *pair-offset*, specify the offset of the \$PAIR in either the MCT or UCT control block with which this table is to be associated. If the table pair in the MCT or UCT is not defined through the \$PAIR macro, an assembler MNOTE is issued.

Specify TABLE=END to terminate a scan table.

Other operands are ignored if TABLE is specified, and a label is required on the \$SCANTAB macro if a table is being started. If TABLE= and NAME= are both not specified, only the mapping of an \$SCANTAB entry is generated by the macro.

#### **NAME=**

Specifies a character name for the scan table entry that indicates the scan keyword being defined. For example, PRINTER, CONSOLE, or JOENUM parameter on the OUTDEF statement. If TABLE= and NAME= are both not specified, only the mapping of an \$SCANTAB entry is generated by the macro.

#### **CONV=**

Specifies the conversion to do (and defines the valid input) when the keyword defined by NAME= is encountered during a scan. CONV= is required if \$SCANTAB is used to generate a table entry. The following specifications are valid:

##### **LONG**

Indicates that if you also specify DISPALL=LONGONLY on this macro call, this call displays on a DISPALL= request.

##### **NUMxxxx**

Indicates that the keyword represents a binary value. The optional second positional defines a multiple to round the input value to before storing. The optional third positional defines a value by which the value of the keyword is multiplied before storing. For example, CONV=(NUM,8,100).

The xxxx can be specified to the following values:

##### **T**

Indicates that a thousands separator (,) is used to display the numeric value. For example, 1,234 instead of 1234.

##### **U**

Indicates that the value in the field is treated as an unsigned integer.

##### **S**

Indicates that the value in the field is treated as a signed integer.

##### **\***

Indicates that an asterisk (\*) is displayed when the value in the fullword numeric field is X'FFFFFFFF'.

#### **HEX**

Indicates that the keyword represents a hexadecimal value that is stored in binary. The optional second and third positionals operands are the same as CONV=NUM.

#### **CHARxxxx**

Indicates that the keyword represents a character value and defines the allowed character input. If the first positional for CONV= is CHAR, any characters that are not required for syntax within \$SCAN are valid for the value unless a specific list of characters is provided.

CONV=CHAR allows right and left parenthesis as input for the scan keyword defined by the \$SCANTAB entry. CONV=(CHAR,x1,x2,x3,...,xn) where x1-xn are specific characters allowed to be specified for the keyword defined by the \$SCANTAB entry and allows right and left parenthesis if they are in the character list (defined by x1-xn).

The rules for coding parenthesis are as follows:

- A right parenthesis is accepted by itself.

- A left parenthesis, if specified, must be part of a balanced parenthesis pair.

If the first positional is CHARxxxx, for a 1- to 7-character string xxxx, the input must fall within the character sets defined by the xxxx or be one of the specific list of characters that may be optionally provided as described below.

The xxxx can be specified to the following values:

**A**

Indicates the alphabetic character set A-Z.

**F**

Indicates that the first character of input must be alphabetic.

**G**

Indicates that the characters \* and ? are valid.

**H**

Indicates the hexadecimal numbers 0 through 9 and characters A through F. The value is right-justified and padded with zeros unless A, S, F, J or additional positional operands are specified.

**J**

Indicates the first character of input must be alphabetic or special national (JCL rules), even though the remaining input might have less strict input rules.

**N**

Indicates the numerics 0 through 9. The value is right-aligned and padded with zeros unless A, S, F, J or additional positional operands are specified.

**R**

Indicates the character data must be right-aligned rather than left-aligned.

**S**

Indicates the special nationals \$, @, and #.

**Z**

Indicates the pad character is X'00'.

A specific list of allowed characters might be specified as the second, third, and so on, positional operands for CONV. They might be specified as single characters, or as 2-character hex values. The following is a valid example:

```
$SCANTAB  NAME=CONCHAR,  CB=HCT,  FIELD=$CCOMCHR,  CONV=(CHAR,.,-
[ ,4D,+,|,50,!,$,*,^,-,/,%,_,?,:,#,@,=,"),
RANGE=(L'$CCOMCHR,L'$CCOMCHR),
CALLERS=($SCIRPL,$SCIRPLC,$SCDCMDS),
PSTSCAN=(PSTCNCHR,SET)
```

**STCKxxx**

For displays only. Indicates that the field contains a time stamp in STCK/STCKE format that should be displayed in the format (yyyy.ddd,hh:mm:ss).

The xxx indicates additional options regarding the format of the field and the display. It can be specified to the following values:

**E**

Indicates that the field is a STCKE not a STCK.

**U**

Default display but without parenthesis.

**L**

Indicates a Long display in the format day mon year AT hh:mm:ss.

**W**

Includes day of week (Long display only) dayname day mon year AT hh:mm:ss.

**M**

Displays time to minute precision only.

## Z

Indicates that the STCK value must be converted to local time.

## 0,1,2,3,4,5,6

Indicates the number of decimal places of precision the second should be displayed to. This specification is not honored for 4-byte fields.

## A

Indicates that the actual value to be displayed or filtered is not the time stamp itself but the difference (in seconds) between the STCK value and the current time. The same as CONV=NUM, the second and third positionals are scaling and rounding values. For example, CONV=(STCKA,,60) gives minutes.

## T

Indicates that the age must be displayed with a thousands separator (.). It is used together with A.

**Note:** For SETs, the prescan and postscan routines are given control; otherwise, there is no effect.

## FLAG[,LIST][,n]

Indicates the keyword that represents a flag value stored as the setting of one or more bits within a single flag byte. The allowed values and associated bit settings are defined by VALUE=.

## LIST

An optional second positional parm to specify that JES2 should list all matching values not only the first.

## n

An optional third positional parm that specifies the number of triplets in the VALUE= list that JES2 displays. (JES2 scans all other parameters for SET and FILTER calls.)

## ALIAS

Indicates that the keyword is the alias of another keyword and SCANTAB= specifies the label of the scan table entry mapping that other keyword.

## VECTOR

Indicates that the keyword represents a vector of values. Another level of scan is used to process the vector of values that is specified within parentheses. Therefore, each value is defined positionally by another scan table that SCANTAB= points to.

## SUBSCAN

Indicates that the keyword requires another level of scanning to scan suboperands. A completely recursive level of scanning is done and the keyword represents any types of suboperands that CONV= specifies. SCANTAB= specifies the address of a doubleword containing the addresses of two scan tables to pass to the recursive \$SCAN call.

## CB=

Specifies one of the 'primitive' control blocks known by \$SCAN as the control block containing the fields representing the value of the keyword, or as the starting point for a control block search for that field. If a control block is not found for a keyword during a scan, and a PRESCAN routine for the keyword does not then supply the control block address, \$SCAN issues a \$ERROR. CB= is required if CONV= is not specified as SUBSCAN, VECTOR, or ALIAS unless PRESCAN= is specified.

## HCCT

Indicates the JES2 HCCT control block.

## DTE

Indicates the current daughter task element (DTE) at the time of the scan.

## TOKEN,name|TOKENname,HOME| TOKENnamePRIMARY|TOKENnameSYSTEM|

## TOKENnameSUBSYS

Specifies the NAME associated with a name/token pair (created using the \$TOKENSR service) which contains the address of the control block that contains the fields. NAMES can be up to 16 bytes long, and must match the name specified on a \$TOKENSR call. The first 4 bytes of the token are assumed to be the control block address. The remaining 12 bytes of the token are not used.

The control block address (from the token) is combined with the dsect name specified in DSECT= and field name to determine the field address. Token name on second operand of CB= is required and only allowed, if TOKEN is specified on first operand of CB= keyword.

NAME/TOKEN pairs can be created at various levels. JES2 supports tokens at the SYSTEM, TASK, or HOME level. The third operand on CB= specifies the level passed on the \$TOKENSR service call. If the third operand is not specified, it defaults to TASK level.

**HCT**

Indicates the JES2 HCT control block.

**PCE**

Indicates the current processor control element (PCE) at the time of the scan.

**DCI**

Indicates a JES2 device control table (DCI), found by scanning all the DCIs for one whose DCIDEVN field corresponds to the NAME= specified and the device number found during the scan.

**UCT**

Indicates the installation-defined user control table (UCT), which is pointed to by the \$UCT field of the HCT.

**TEMP**

Indicates a temporary control block should be allocated with a length defined by the second positional operand.

**size**

Indicates the length (in bytes) of the control block

If ACTFLAG is defined, \$SCAN does not determine the activity level of the control block.

**PARENT**

Indicates the control block determined at the scan level 'above' this scan level should be used, that is, when a control block is found for a CONV=SUBSCAN or CONV=VECTOR keyword, that control block is the parent control block for the resulting subscanning.

**PRESCAN**

Indicates the routine where the control block is located.



**Attention:** \$SCAN does not do any searching or validating before the prescan.

If ACTFLAG is defined, \$SCAN does not determine the activity level of the control block.

**NOCB=**

Specifies the action to take if JES2 couldn't find a control block (that is, a control block address of zero.)

**FAIL**

\$HASP003 is issued with text REQUIRED CONTROL BLOCK(S) NOT AVAILABLE. NOCB=FAIL is the default.

**SKIP**

The \$SCANTAB is skipped.

**Note:** Prescan routines, if any, locate the control block before checking for a zero value.

**CBIND=**

Specifies how to find the control block required for this keyword, if the primitive control block is not it. The search starts from the primitive control block address, and performs a series of operations of fields within each control block along the way. The fields used are defined by the first and second operands and the operation is defined by the third operand in each of a set of operand triplets defined to CBIND=. CBIND must be specified as a list consisting of a multiple of these three operands.

**field-name1**

Identifies the name of the control block field that is to be compared to the subscript name, until a match is found.



**dsect1**

Defines a DSECT name for the control block that contains the field.

**instruction1**

Defines the operation to be performed, with the current control block address, against the field.

The allowed operations are L (load), LA (load address), A (add), AH (add halfword), AL (add logical), S (subtract), SH (subtract halfword), and SL (subtract logical). If the operation specified is preceded by an asterisk (for example, \*LA), then any subscript indexing for the control block search is done before this CBIND operation, rather than after all CBIND operations. Subscript indexing is defined by the SUBSCRIP= operand.

**CLEANUP=**

Specifies routines that should be given control to clean up resources obtained by a prescan routine, even when a postscan routine does not get control (such as an \$SCAN error, filter mismatch). The syntax of the keyword is:

CLEANUP=(rtn,xxx,rtn,rtn,xxx,...)

Where rtn is a routine that is to be called and xxx is optional and can be one of 'HCT', 'HCCT', or 'CR11', indicating the value to be placed in register 11 on entry to the routine. Specify CR11 if the value of register 11 at the time of the \$SCAN call is to be used. If none of HCT, HCCT, or CR11 is specified, then a default is used. The default depends on the location of the routine address: PADDR, UPADDR, SXADDR OR USXADDR: HCT CADDR, UCADDR: HCCT A-CON OR V-CON: HCCT in user env: CR11 otherwise.

**CMDRDIR=**

Specifies the response for the command is subject to redirection. The name specified (cccc) is the 4-character identifier of a redirection group which must correspond to the redirection group name (NAME=) on a \$RDRTAB macro. If the CMDRDIR parameter is not coded, the command is not subject to redirection.

**COMAUTH=**

Specifies the authority required to issue the specified command. Multiple authority levels may be coded for this keyword. Values allowed for this keyword are:

**JOB**

Job authority required.

**SYSTEM**

System authority required.

**DEVICE**

Device authority required.

**COMENT=**

Specifies the qualifier for command authorization. This is combined with the qualifier on the \$COMTAB for the 'command verb' to provide the complete resource name. For example, if the \$T command specifies 'MODIFY' as a COMENT and the \$SCANTAB macro for PRINTER specifies 'DEV' as a COMENT, then the complete resource name is 'MODIFY.DEV'. Note that COMENT= on \$SCANTABs which are not at the highest level of SCAN are ignored.

**COMPMSG=**

Specifies whether (YES) or not (NO) JES2 should display the \$HASP894 DISPLAY COMPLETE after the \$D command processing completes. **A second parameter, which is optional, is used to specify the \$BLDMSG id of a different message to be displayed when completed.**

**COMREJ=**

Specifies under which circumstances commands should be rejected. The value allowed for this keyword is:

**REMOTE**

Rejects command if issued from a remote.

**DELTEXT=**

Specifies the text that JES2 appends to an element's display on a DISPLAY/DELETE request.

**DEFAULT**

Indicates that JES2 appends the default text, - ELEMENT DELETED on a DISPLAY/DELETE request.

**NONE**

Indicates that JES2 will not append any text on a DISPLAY/DELETE request.

**DISPALL=**

Specifies if this keyword is displayed on a display-all subscan request.

**YES**

Displays the keyword.

**NO**

Does not display the keyword.

**LONGONLY**

Displays the keyword only when you specify CONV=LONG on this macro call or LONG=YES on the \$SCAN call.

**NOTLONG**

Displays the keyword only when the display-all is due to specifying CONV=LONG on this macro call or LONG=YES on the \$SCAN call.

**FILTER=**

Specifies the type of filtering JES2 uses when processing this keyword.

**(YES[ALWAYS,EQ,NEQ,GTLT,NODELIM,VORDER,NOVORDER,NOSET, NOGENERIC])**

Specified as a filter like YES but with any of the following positionals:

**ALWAYS**

This keyword is always a filter when specified on a SET call. When ALWAYS is not specified on a SET call, you must precede it by a division slash (/) to be a filter.

**EQ**

A valid filter delimiter of equal (=).

**NEQ**

A valid filter delimiter of not equal (≠ or <>).

**GTLT**

Valid filter delimiters other than equal or not equal are allowed.

**NODELIM**

This keyword is accepted as a filter when no filter delimiter is specified. This results in a match on a null VALUE= parameter.

**VORDER**

For CONV=VECTOR keywords, indicates that the vector elements must be in the same order as specified on the filter for a match to occur.

**NOVORDER**

For CONV=VECTOR keywords, indicates that the vector elements may occur in any order.

**NOSET**

Indicates that the filter is not allowed on a SCAN=SET call.

**NOGENERIC**

Generic characters \* and ? are not to be treated as generics and must match exactly on character comparisons.

If EQ, NEQ, or GTLT is not specified, the default is as if all three were specified. If CONV=FLAG, the default is as if EQ and NEQ were specified.

**NO**

If you specify FILTER=YES on the \$SCANTAB for a command, you can specify one or more selection criteria on the display command, and only when the parameter is equal to the value specified will the elements be displayed. If you specify FILTER=NO on the \$SCANTAB for a command, then you can only specify one parameter on a display command. If you specify any other selection criteria, you receive an error message.

**SUBFLD=**

Specifies where to find the search argument or field to match with the numeric subscript.

**field**

Identifies the field in each control block in the chain that is compared to the numeric subscript, until a match is found.

**dsect**

Specifies the name of the DSECT for the control block.

**length**

Specifies the length of the field to be compared.

**SUBSCR=**

Specifies an allowable subscript range for the input specifying this keyword. If SUBSCR is specified, the allowable input forms for \$SCAN are 'keyword(subscript)' and 'keywordsubscript'. SUBSCR is specified as a list of 2, 3 or 4 values, with the first being the lowest allowed subscript value and the second being the highest allowed value. The first and second operands must be numeric values with one exception; single character alphanumeric subscripts can be used, with 'A' corresponding to value X'C1', '4' to value X'F4', and so on.

The optional third value specifies an index value optionally used during the search for the control block for this keyword. After (by default) or during the CBIND processing in that search, the subscript value is used to index into the current control block to find the correct sub-block for the keyword. The lowest subscript is assumed to correspond to the 0th sub-block and the length of each sub-block is defined by the third positional value of SUBSCR.

The optional fourth value specifies that the "high" and "low" values are to be used as offset values into the HASP (HCT) or USER (UCT) control tables.

**FIELD=**

Specifies the name and length of the field associated with the keyword value in the specified control block. The field must be within the DSECT specified by DSECT=, or must be an absolute offset if DSECT=0 is specified. The length is defined by the second positional parameter, and defaults to the assembler-defined length of the field label. FIELD= is required unless CONV= is SUBSCAN, VECTOR or ALIAS.

**DSECT=**

Specifies the DSECT name required to resolve the field specified by FIELD= in the control block found by the \$SCAN search. If FIELD= is specified as an absolute offset into the control block, DSECT should be specified as DSECT=0.

**RANGE=**

Specifies the allowed range for the input. For keywords for which CONV is NUM, HEX, or CHARN RANGE specifies a binary range. For keywords for which CONV is CHARxxxx, for CHARxxxx not equal to CHARN, RANGE specifies a length range. RANGE and VALUE are mutually exclusive.

**RELATED=**

Specifies a list of related \$SCANTAB entries. You should use this parameter when you modify a \$SCANTAB parameter that directly affects the value of another \$SCANTAB call.

**SSOPT=**

Specifies whether (YES) or not (NO) a subscript specification is optional on all \$SCAN call types.

For SCAN=DISPLAY calls, the subscript specification is always optional regardless of the value you specified here.



**Attention:** If you specify SSOPT=YES, JES2 assumes a subscript of (\*) when a subscript is not specified.

**VALUE=**

Specifies the allowed specific values a keyword may have. VALUE is used to limit input to only certain values, instead of using RANGE to limit the input to a range of values. RANGE and VALUE are mutually exclusive.

For keywords for which CONV= is not specified as FLAG, this keyword is specified as a list of allowed values. Note that the input must match the VALUE= specification exactly. For example, if the value 000293 is specified as input, it is within the allowed range for RANGE=(66,400), that is between 66 and 400, but it does not match the VALUE=(36,2,99,293,4) specification, exactly.

For CONV=FLAG keywords, VALUE is specified as a list making up a set of triplets of input, that is VALUE=(a1,b1,c1,a2,b2,c2,...). For each set of three operands, as shown, the first (a) is the allowed value the keyword may have, the second (b) is a flag byte setting to 'or' on in the FIELD if the keyword is given this value, and the third (c) is a flag bytes setting to 'and' off in the FIELD. For example, to implement a keyword with values of YES or NO, which is represented by a single flag bit setting specify the following:

```
CONV=FLAG, VALUE=(YES, YESFLAG, FF, NO, 0, FF - YESFLAG)
```

If the keyword has no input value, that is, there is value in the keyword being specified alone, VALUE should consist of one triplet with the first operand null.

### **CALLERS=**

Specifies one or more caller ids (in a parenthesized list) for which this scan table entry is to be used. If CALLERS is not specified, the table entry is used for any \$SCAN caller. This operand is useful, for example, when a scan table is to be used for multiple parameter statement purposes and not all keywords are valid in every case. Note that \$SCAN supports multiple entries specified in a scan table for the same NAME= keyword with different CALLERS= specifications. Valid callers are:

#### **Valid Callers**

**Identifies the:**

#### **\$SCOPTS**

JES2 initialization options (for example, COLD, WARM, REQ, NOREQ)

#### **\$SCIRPL**

JES2 initialization statements

#### **\$SCIRPLC**

console-issued commands during JES2 initialization

#### **\$SCDCMDS**

display commands in HASPCOMM

#### **\$SCSCMDS**

set commands in HASPCOMM

#### **\$SCDOCMD**

short forms of the display commands

#### **\$SCSTCMD**

start commands

#### **\$SCPCMDS**

stop commands

#### **\$SCDDIAL**

dialog display form

#### **\$SCSDIAL**

dialog set form

#### **\$SCECMDS**

reset commands

#### **\$SCACMDS**

add commands in HASPCOMM

#### **\$SCRCMDS**

delete commands in HASPCOMM

#### **\$SCLTCMD**

Output long display

**\$SCECMDA**

RESET COMMANDS (single)

**\$SCZCMDS**

HALT commands

**\$SCHCMDS**

HOLD commands

**\$SCRLCMD**

RELEASE commands

**\$SCCCMDS**

CANCEL commands

**\$SCTOCMD**

\$TO commands

**\$SCCOCMD**

\$CO commands

**\$SCPOCMD**

\$PO commands

**\$SCOCMDS**

\$O command

**\$SCLOCMD**

Output short display

**\$SCLCMDS**

\$L command

**PRESCAN=**

Specifies the name of one or more routines to be entered just after determining the parameter input contains this keyword and before scanning the input any further. The routine does not have to be in the same CSECT as the table. Register 1 points to the scan work area (SCWA) on entry to the routine and the routine can change the SCWA fields and use return codes to direct the actions of \$SCAN. An optional second positional parameter of SET, DISPLAY, FILTER, or DELETE on PRESCAN after each PRESCAN routine name indicates that the PRESCAN routine should be called only for the specified \$SCAN calls. You can specify more than one of these parameters for each routine name.

Values of HCT, HCCT, or CR11 can also be specified as positionals after the routine name to influence the contents of R11 on entry to the routine; these might be required in some multi-environment cases. If not specified, the value of R11 is set as follows:

- HCT, if the routine address is in the PADDR, UPADDR, SXADDR, or USXADDR;
- HCCT, if the routine address is in the CADDR or UCADDR, or if the \$SCANTAB is assembled in the USER assembly environment;
- CR11, if none of the above apply.

Specify CR11 if the value of register 11 at the time of the \$SCAN call is to be used. If none of HCT, HCCT, or CR11 is specified, then a default is used.

**PSTSCAN=**

Specifies the name of one or more routines to be entered after all scanning (including possible subscanning) is done for this keyword. PSTSCAN= and the routine interface are the same as those for PRESCAN=.

**SCANTAB=**

Specifies another one or more scan tables or table entry required when scanning this keyword. For CONV=ALIAS, it specifies the address of another scan table entry defining the keyword of which this keyword is an alias. For CONV=VECTOR, it specifies the address of another complete scan table defining the values allowed for each element of the vector. For CON=SUBSCAN, it specifies the address of a doubleword containing the addresses of two complete scan tables, for example, one user scan table and one JES2 scan table, to be used in the recursive \$SCAN call that performs the subscanning required.

## **\$SCANTAB**

If CONV=VECTOR or CONV=SUBSCAN is specified, the pointer to the next set of scan tables is calculated as an offset into either the MCT or UCT. This is specified by the second positional operand on this keyword. For example:

```
$SCANTAB SCANTAB=(MCTPRTU,MCT) . . .
```

Generates (MCTPRTU – MCT) for the offset into the \$MCT of the scan table pair.

```
$SCANTAB SCANTAB=(UCTPRTU,UCT) . . .
```

Generates (UCTPRTU – UCT) for the offset into the \$MCT of the scan table pair.

```
$SCANTAB SCANTAB=(OWNPAIR,ADDR) . . .
```

Generates (OWNPAIR – ADDR) as the address of the table pair.

```
$SCANTAB SCANTAB=(OWNPAIR,VCON) . . .
```

Generates (OWNPAIR – VCON) as the VCON of the table pair.

### **VCOUNT=**

Specifies the number (1-255) of vector elements this scan table entry defines. VCOUNT is ignored if it is specified for an entry in a scan table that is specified to SCANTAB= for a CONV=VECTOR keyword. It allows a single scan entry to define multiple elements of a vector, with the associated fields for the elements being FIELD, FIELD plus the field length, FIELD plus twice the field length, and so on. The default is a value of 1.

An second positional parameter of IGNORE in VCOUNT, which is optional, indicates that null input for vector elements is allowed and the associated fields should not be changed in any way.

A third positional parameter, which is optional, specifies the size of a vector element. This is the number of bytes that must be added to the current vector element to locate the next element. The default is the size of the field. A value of 0 indicates to use the same field for all elements, for example, a postscan exit is processing the elements.

### **OBS=**

Specifies whether the keyword specified for NAME= is to be considered obsolete (the default is OBS=NO). If OBS is specified as YES, \$SCAN should consider it to be an error if this keyword is found during a scan, but return a less severe return code and message to the caller.

### **MINLEN=**

Specifies the minimum character length of the keyword defined by this \$SCANTAB entry that may be used to reference the keyword in parameter input. For example, if NAME=FORMS is specified, and MINLEN=2, then: FO, FOR, FORM, and FORMS are valid keyword references; F, FOX, and FORMSX are invalid. If MINLEN= is not specified, the valid keyword specification is the entire keyword; no abbreviated forms are allowed.

### **MSGID=**

Specifies the 3 or 4-digit message ID for the \$HASP $nnn$  message identifier that is used when a SCAN=DISPLAY includes a display line in a \$SCAN call. This message ID is ignored by \$SCAN except at the highest level of scanning. For example, it is used for the PRINTER $nn$  statement, but it is ignored for the FORMS= keyword on the PRINTER $nn$  statement.

### **DISPKEY=**

Specifies whether (YES) or not (NO) the keyword name is displayed as part of the \$SCANTAB output. If YES is specified, the keyword specified by NAME= on this macro is displayed with its value. If NO is specified, only the value assigned to the keyword is displayed.

### **TEXT=**

Specifies the text string produced by this \$SCANTAB call if SCAN=(DISPLAY,ALL) is specified on the \$SCAN macro instruction. You can specify a character string up to 255 characters; enclose the string in single quotation marks.

**Note:** If TEXT= is specified, do not also specify NAME= or TABLE=.

**TOKEN=**

Specifies the address of a control block that is passed to the \$SCAN routines, PRESCAN, POSTSCAN, and DISPLAY.

**DISPLAY=**

Specifies whether (YES) or not (NO) the value assigned to a keyword is displayed as part of the \$SCANTAB output. If YES is specified, the value associated with the keyword specified by NAME= on this macro is displayed. If NO is specified, the value associated with the keyword specified by NAME= on this macro is not displayed.

**CRLF=**

Specifies whether (YES) or not (NO) the line of message text added by this SCANTAB call is displayed following a carriage return and line feed (that is, displayed on a new line) during display processing. The default, NO, specifies the additional text is appended immediately after the existing message text. A second optional value is the line type for multi-line WTOs. Valid values are C, L, D, DE, and E. These are only valid if CRLF=YES. See the WTO macro for complete descriptions.

**ROUT=**

Specifies the route codes to which this message is to be routed. These values are passed to the display routine for processing. This route code is ignored by \$SCAN except at the highest level of scanning. If you provide more than one code, separate each by a comma and enclose the list in parentheses.

**DESC=**

Specifies the descriptor codes for this message that are passed to the display routine for processing. This descriptor code is ignored by \$SCAN except at the highest level of scanning. If you provide more than one code, separate each by a comma and enclose the list in parentheses.

**JESROUT**

Specifies whether (YES) or not (NO) the JES specific route code of 42 is to be used. The default is NO. It is only used by \$BLDMSG and ignored by \$SCAN.

**DISPERS=**

Specifies a list of flag bits that must be set on in the flag byte specified on the DISPER= keyword for the call to the SCAN macro if this SCANTAB is to be used. If the required bits are not set on, this SCANTAB is not used.

**ACTDISP=**

Specifies the activity condition required for display of this keyword. ACTIVE indicates display only in the case of activity; INACTIVE indicates display in case of inactivity; and BOTH indicates display regardless of activity. The ACTFLAG bit is used to determine whether or not there is activity. ACTDISP=BOTH is the default.

**ACTFLAG=**

Defines a field in the control block pointed to by CB, CBIND, and SUBSCRIP= that is used to determine if there is any activity on the logical (for example, node) or physical (for example, printer) device. If any bits are on (set to 1), activity is assumed and the ACTSET= keyword determines if a set-type \$SCAN is permitted.

If CB=PRESCAN or CB=TEMP is specified, \$SCAN ignores the ACTFLAG parameter.

**ACTSET=**

Specifies the activity condition required for a set-type \$SCAN call. ACTIVE indicates activity is required; INACTIVE indicates that inactivity is required; and BOTH indicates a set-type \$SCAN call is allowed regardless of activity. The ACTFLAG bit is used to determine whether or not there is activity. ACTSET=BOTH is the default.

**GENSET=**

Specifies whether (YES) or not (NO) a set-type \$SCAN call is permitted for generic requests. A generic request is one that includes an asterisk (\*) within the symbolic subscript, for example, PRT(99-\*). GENSET= does not affect the processing of range requests, for example, PRT(99-999). GENSET=YES is the default.

**LKUPFLD=**

Specifies an argument used to locate a control block. As part of a CB= or CBIND= search, the LKUPFLD= specification is used to match the specified symbolic subscript. If CB= and CBIND= are not used, LKUPFLD= is used as the search argument for the control blocks defined by SUBSCRP=.

**SCAN=**

Specifies the call types that can be used by \$SCAN to call \$SCANTAB. The list of valid calls is all the scan call work area (SCWA) equates from the SCWATYPE field, for example SCAN=SCWASET+SCWADISP+SCWACR allows set-, display-, and create-type \$SCAN calls. If SCAN=SCWASET+SCWACR is specified, display-type calls are invalid. If SCAN= is not specified, all call types are allowed.

**WARNING=**

A one-byte warning mask that indicates when this scan table entry should be halted and a warning-level diagnostic message sent to the caller. This warning mask is compared with the mask specified by WARNMSK= on the \$SCAN macro call. If any bits match, the SCAN process will be halted.

**JOB=YES | NO**

Specifies whether a display of this table entry by a subsequent \$SCAN macro is to include a job identifier. Code this parameter only on high level \$SCANTAB entries.

**YES**

Include the job identifier in the display.

Code YES only if you are defining a table entry that modifies an IBM-defined command. For a list of the IBM-defined commands that you can modify through the \$SCANTAB macro, see [Table 10 on page 450](#). If you code YES, you must have previously set PCE field PCEJQE to point to the JQE that corresponds to the JOE to be displayed.

**NO**

Do not include the job identifier in the display. This is the default.

**FILTER\_XCLUDE=**

Specifies a list of \$SCANTABs that represent filters. The filters are mutually exclusive with this filter keyword.

**WIDTH=nnn**

Indicates the number of places in the output area (from 1 to 32) to be used to display the results. This is only valid with CONV=NUM (right justified), CONV=HEX (left justified), or CONV=CHAR (right or left justified). This parameter is optional.

A second operand on width, PAD0, (valid only for CONV=NUM) indicates to pad the number with leading 0s.

**COMPCE**

Specifies the identifier (sequence number) of the command processor (PCE) that processes the command defined by this table. The default value is \$CMDNORM (0).

**Note:** COMPCE= on \$SCANTABs that are not at the highest level of scan is ignored.

**FORMAT**

Specifies a list of format options to be used when displaying the output of this request. Options are listed in parenthesis, which is separated by commas.

**Note:** FORMAT= on \$SCANTABs, which are not at the highest level of scan, are ignored.

Valid options are:

INDENT|NOINDENT - Lines after the title line being indented.

MSGID|NOMSGID - Lines after the title line that includes a message ID.

Default is FORMAT=(INDENT,MSGID)

## Environment

- JES2 main task or during initialization and termination.

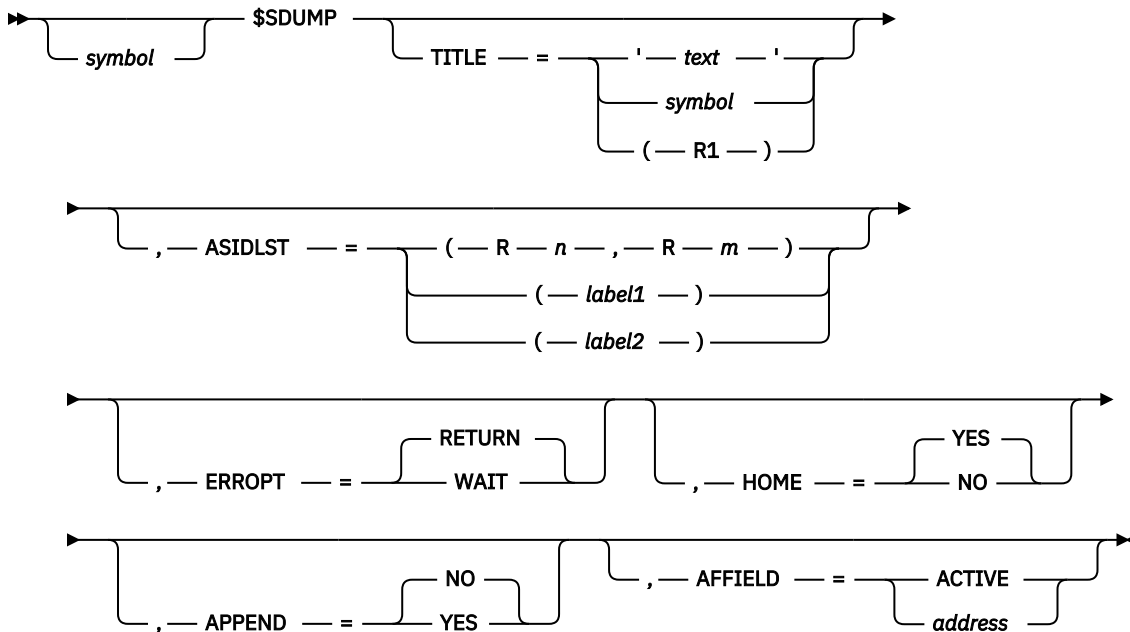


- \$WAITS can occur.

## \$SDUMP – Take an SDUMP of storage

Use \$SDUMP to dump the storage of selected address spaces.

### Format description



#### TITLE=

Specifies the title of the dump. You can specify the title of the dump as straight text within quotation marks or you can supply a symbol that identifies the beginning of the textual title or you can supply a register whose contents is the address of the textual title. If you supply a symbol or register, the symbol or register must point to a one byte length field followed by the text. If TITLE is not specified a default title for the dump is used.

#### ASIDLST=

Specifies a list of asids (up to two) associated with the address spaces to be dumped besides the home address space if HOME=YES. Label1 and label2 must define halfwords that contain the asids. Rn and Rm are two different registers that contain the asids in the right-most half of each register. The left-most half of each register must be zero.

#### HOME=

Specifies whether the home asid is dumped. HOME=YES is the default indicating that the home asid is to be dumped.

#### ERROPT=

Specifies the action to be taken should the dump fail. ERROPT=RETURN indicates that when the dump fails, return to the caller should take place. ERROPT=WAIT indicates that a WTOR is to be issued to the operator and the \$SDUMP processing is to wait for an appropriate reply. ERROPT=RETURN is the default.

#### APPEND=

Specifies whether the title supplied with TITLE= is to be appended to the default title. APPEND=NO is the default and indicates that the title supplied is not to be appended to the default title.

#### AFFIELD=

Specifies the address of the affinity field that identifies the members to be dumped or 'ACTIVE' meaning use XMAMEMUP (all active members of the MAS). If not specified, only this member is dumped.

## Environment

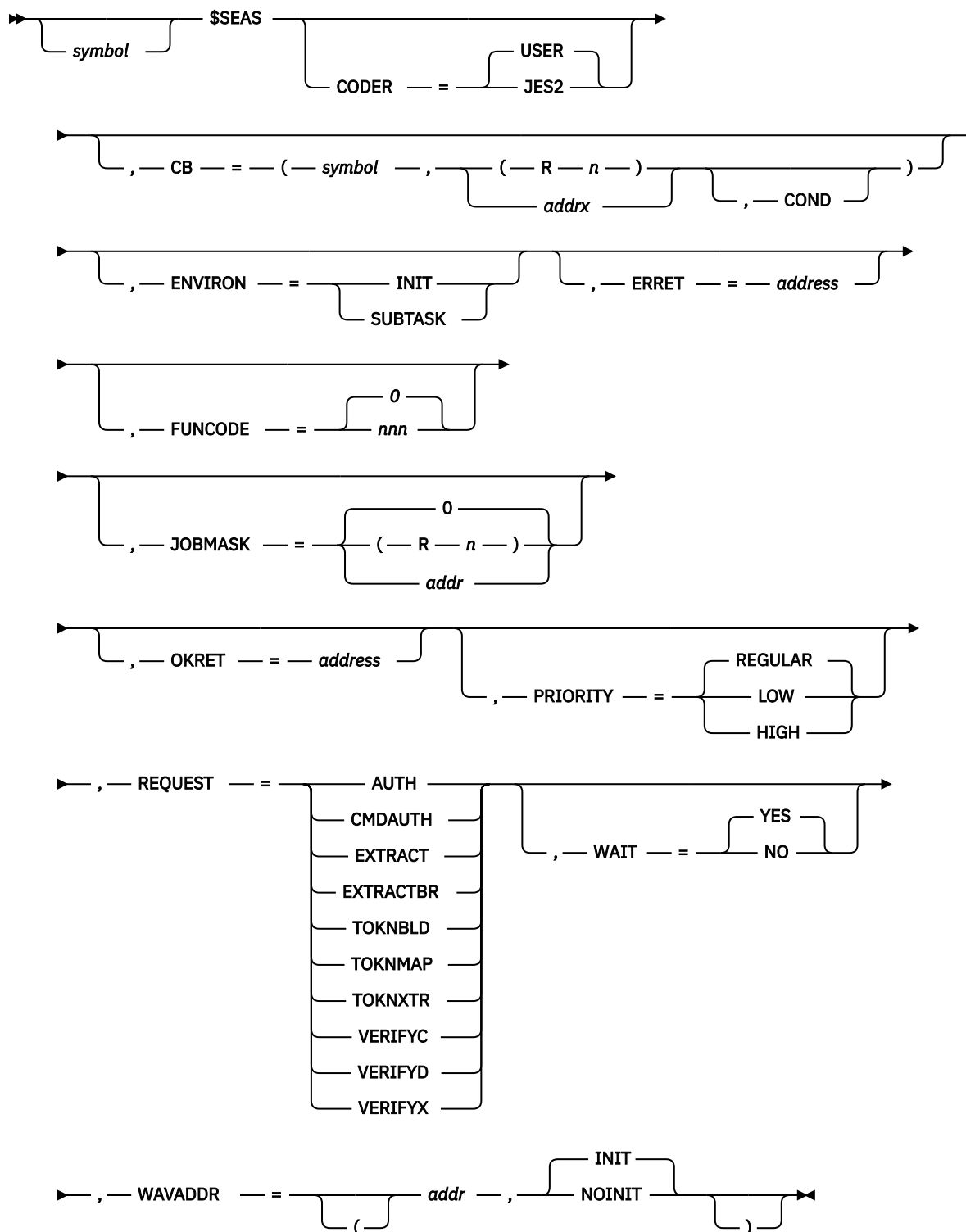
- Main task.
- MVS WAIT can occur (if ERROPT=WAIT).

## **\$SEAS – Security authorization services**

---

The \$SEAS macro is the JES2 interface to the security authorization facility (SAF). The macro determines the environment invoking the macro and then either calls SAF directly or invokes a service routine to call SAF. Before passing control to SAF, JES2 invokes Exit 36. Before control returns to your routine, JES2 invokes Exit 37.

## Format description



### **CODER=JES2|USER**

Specifies whether IBM-supplied or installation-written code is invoking this macro.

**Default:** USER

### **CB=**

Specifies the ID and address of the control block \$SEAS places in the \$WAVE. The parameters for CB= are:

**symbol**

A symbolic that points to the 4-byte name of a JES2 control block for this request.

**addrx|(Rn)**

The address or a register that contains the address of the control block specified in symbol.

**COND**

This parameter is allowed only when the second parameter is a register. JES2 verifies that the register specified contains an address before updating the \$WAVE with the control block name and address.

**ENVIRON=INIT|SUBTASK**

Specifies whether this call is being made from a subtask (SUBTASK) or during initialization (INIT). Code this parameter only if you issue \$SEAS from either of these two environments. If invoking this macro from a subtask, you **must** specify ENVIRON=SUBTASK to override the assembly environment of the module.

**Note:** If you do not specify ENVIRON=, the default is the current assembly environment.

**ERRET**

Specifies the label of, or a register that contains, the address of a routine that receives control if the RC is non-zero. ERRET= is optional, and either a label or a register notation can be used to specify the error routine address.

**FUNCODE=0|nnn**

Represents the location and/or type of call. When you specify CODER=USER, this parameter is optional and can be between 0 and 255. IBM-supplied routines specify values of 1 to 21 for this parameter, so you should avoid these codes unless you create calls similar to those supplied. Your values for FUNCODE= should start at 255 and work downward to avoid conflicting with the IBM-defined values. Exits 36 and 37 use this code to determine the location and type of call.

When you specify CODER=JES2, this parameter is required and must be between 1 and 20. The meanings of FUNCODE= in IBM-supplied routines are:

Decimal Value	Meaning	Related Control Block
0	Reserved for user code	
1	Initialize security environment	SFI
2	Security environment create	JCT
3	Security environment delete	JCT
4	Extract security information for this environment	SJB
5	SYSIN data set create	IOT
6	SYSOUT data set create	IOT
7	SYSIN data set open	SDB
8	SYSOUT data set open	SDB
9	Process SYSOUT data set open	SDB
10	Process SYSOUT data set select	PSO
11	TSO/E cancel	JCT
12	Command authorization	none
13	Printer data set select	PDDb
14	Data set purge	IOT
15	Notify user token extract	None

Decimal Value	Meaning	Related Control Block
16	Token build	SFI
17	RJE signon, NJE source for command authorization	SWEL
18	Device authorization	PCE
19	NJE SYSOUT data set create	SFI
20	Reserved	None
21	Reserved	None
22	Update of JESNEWS	SJB
23	Build JESNEWS token	IOT
24	Subtask to create access control environment element (ACEE) for general subtasks	None
25	Audit for job in error	None
26	Authorization for \$DESTCHK	DCW
27	SYSOUT data set create for trace.	IOT
28	SYSOUT data set create for system job data sets (for example, JOBLG)	SFI
29	SYSOUT data set create for JESNEWS.	IOT
30	Transmit or offload of SYSOUT.	PCE
31	VERIFYX for receive or reload of SYSOUT.	SFI
32	Transmit or offload of job.	PCE
33	Reserved	None
34	Spool browse data set open	SDB
35	\$SEASFS - Scheduler service, TOKNXTR - SSW	SSW
36	\$SEASSWM - SWM modify ALTER AUTH	None
37	\$SEASAPI - SYSOUT application programming interface	None
38	\$SEASCLA - SECLABEL affinity extract	JQE
39	\$SEASCLE - DCT SECLABEL extract	DCT or NIT
40	\$SEADIRA - Seclabel dominance	None
41	\$SEASPLR - SPOOL I/O AUTH check	None
42-255	Not in use	Not in use

**JOBMASK=0 | (Rn) | addr**

Specifies the address, or a register that contains the address, of the JOBMASK used by exits 36 and 37, if needed. This parameter is optional.

**Default:** 0

**OKRET**

Specifies the address of the routine that receives control when the RC is zero. You can specify a label that corresponds to the routine's address, or a register that contains the address of the routine.

**REQUEST=AUTH | CMDAUTH | EXTRACTBR | TOKNBLD | TOKNMAP | TOKNXTR | VERIFYC | VERIFYD | VERIFYX**

This required parameter specifies the type of request passed to SAF. These requests correspond to the RACROUTE macro requests: AUTH, EXTRACT BRANCH=NO, EXTRACT BRANCH=YES, TOKENBLD, TOKENMAP, TOKNXTR, VERIFY ENVIRON=CREATE, VERIFY ENVIRON=DELETE, and VERIFYX. CMDAUTH corresponds to the MVS console command authorization macro. An explanation of these requests appears in *z/OS Security Server RACF Macros and Interfaces* and *z/OS MVS Programming: Assembler Services Reference ABE-HSP*.

For EXTRACT and EXTRACTBR, the address of the information returned by the RACROUTE TYPE=EXTRACT request is stored in WAVEXTLA.

**PRIORITY=LOW | REGULAR | HIGH**

Specifies the priority of the request. Reserve using high priority for requests that need the best performance. For example, validation of a real-time transaction probably deserves high priority. A small batch job opening a SYSIN data set, does not. PRIORITY= is valid only in the JES2 main task.

**Default:** Regular

**WAIT=YES|NO**

Specifies at what point in its processing \$SEAS is to return. WAIT=NO queues the request and returns immediately to the caller. This is useful when initializing a work access verification element (\$WAVE).

If you specify WAIT=YES and the address of a previously initialized \$WAVE, \$SEAS waits until the request is processed.

WAIT= is valid only in the JES2 main task.

**WAVADDR=****addr**

Specifies the address of the Work Access Verification Element (\$WAVE).

**INIT**

Initializes the \$WAVE fields with the values specified on this macro and instructs SAF to begin its processing.

**NOINIT**

Does not initialize the \$WAVE. If you specify NOINIT, you must initialize the appropriate fields in the \$WAVE. Normally, use NOINIT on a second \$SEAS call after initializing the \$WAVE.

If you code NOINIT, specify only the \$WAVE address and ENVIRON=.

**Return codes**

The following return codes (in decimal) are returned in register 15.

**Return Code****Meaning****0**

Processing successful (no errors). SAF granted authorization for access.

**4**

SAF was unable to make an authorization decision.

**8**

SAF denied authorization for access or a subtask failure prevented SAF from making an authorization decision.

**12**

\$SEAS could not fulfill a \$GETWORK request for a \$SQD. This call was made with a WAIT=NO; calls that can tolerate a WAIT=YES will not receive a return code of 12.

## Usage notes

See [Appendix E, “Invoking the security authorization facility \(SAF\),” on page 487](#) for a brief discussion on verifying access to resources and using \$SEAS to invoke SAF.

Users of the \$SEAS macro are responsible for setting up the RACROUTE parameter list in the WAVE. The \$SEAS service will add to the parameters passed, the SUBSYS= keyword. The value passed in for SUBSYS consists of the JES2 subsystem name concatenated with the three character JES2 Version number (for example, JES2313). Users of \$SEAS must either specify DECOUPLE=YES (if supported by their security product) or define the SUBSYS value to their security product.

## Environment

- All environments.
- \$WAIT can occur.
- MVS WAIT can not occur. (However, if JES2 was unable to attach any general purpose subtasks during initialization, an MVS WAIT will occur.)

## \$SEPPDIR – Create a user peripheral data information record (PDIR)

---

Use \$SEPPDIR to send a PDIR to an output device immediately before sending a separator. The PDIR is a required control record that is sent to a SNA/RJE remote that is using its spooling capability to allow data set printing. The PDIR record is used to describe the data set (every output record, separator pages, and cards) being sent. If no separator is being sent, do not use this macro instruction. JES2 sends a PDIR preceding the print header and trailer separators. Also, JES2 sends a PDIR preceding a punch separator; no PDIR is sent following a punch file. This macro supports the separator exits (Exit 1 and Exit 15) in modules HASPPRPU. It is not used for the FSS separator exit (Exit 23) in module HASPFSSM.

## Format description



The specified register contains the address of a JES2 buffer.

## Environment

- Main task.
- \$WAIT can occur.

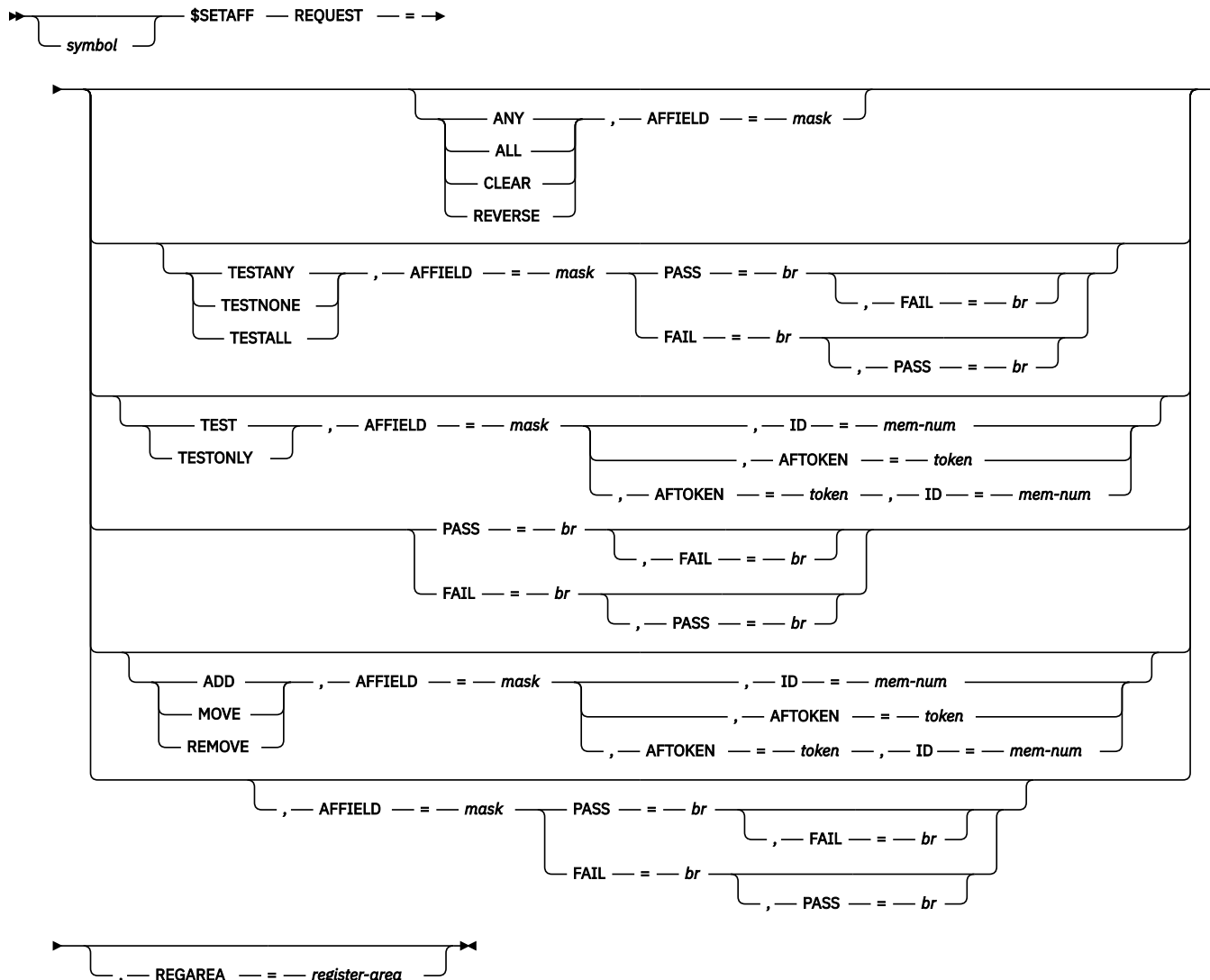
## \$SETAFF – Set \$SIDAFF into correct affinity

---

The \$SETAFF macro generates in-line code to manipulate the data areas related to system affinity. These data areas are:

- The complete system affinity mask, where each bit in the mask represents a unique system.
- The system id number.
- The system affinity token (for a definition of the format of a system affinity token, see the description of the AFTOKEN parameter).

## Format description



### AFFIELD=

The full affinity field that is to be acted upon. The specification can be an RX-type address or a registers (2-12).

### AFTOKEN=

The affinity token to be used. The specification can be an RX-type address or a register (2-12). The format is a 1-byte mask, plus a 2-byte offset into entire mask.

### mask

The single system affinity bit of a particular system.

### offset

The offset to the byte in the full affinity field where the mask bit resides.

### ID=

A numeric system number that is to be converted to the affinity token format. The affinity token format is either be returned in the AFTOKEN field or used to perform the requested function on the affinity field. The specification can be an RX-type address or a register (2-12).

### FAIL=

Specifies a label to be branched to or a register to be branched on if the test requested by either REQUEST=TEST, REQUEST=TESTANY, REQUEST=TESTALL, REQUEST=TESTNONE, or



REQUEST=TESTONLY is false. Either this parameter or FAIL (or both) MUST be specified for REQUEST=TEST|TESTANY|TESTALL|TESTNONE|TESTONLY.

**PASS=**

Specifies a label to be branched to or a register to be branched on if the test requested by either REQUEST=TEST, REQUEST=TESTANY, REQUEST=TESTALL, REQUEST=TESTNONE, or REQUEST=TESTONLY is true. Either this parameter or FAIL (or both) MUST be specified for REQUEST=TEST|TESTANY|TESTALL|TESTNONE|TESTONLY.

**REGAREA=**

An area where registers 0,1,14,15 are to be saved and restored from by this macro. It must be an RX-type address. Registers are not valid.

**Attention:**

This macro might destroy the contents of R0, R1, R14 and R15 unless you provide the REGAREA= keyword.

**REQUEST=**

Requested function to be performed.

**ADD**

Add the system identified by either AFFIELD or AFTOKEN to the specified affinity field. The required parameters are AFFIELD and either ID or AFTOKEN.

**Note:** If both ID= and AFTOKEN= are specified for a request of ADD, REMOVE, TEST, or MOVE, the ID is first converted into a token and placed in the field pointed to by AFTOKEN=. After this conversion, the requested function is performed.

**ANY**

Set an affinity of ANY in the field passed. The required parameter is AFFIELD.

**ALL**

Set an affinity of ALL. The required parameter is AFFIELD. ALL is a synonym of ANY.

**CLEAR**

Clear out the affinity field as AFFIELD.

**MOVE**

Set the affinity to the one system represented by ID or AFTOKEN. The required parameters are AFFIELD and either ID or AFTOKEN.

**Note:** If both ID= and AFTOKEN= are specified for a request of ADD, REMOVE, TEST, or MOVE, the ID is first converted into a token and placed in the field pointed to by AFTOKEN=. After this conversion, the requested function is performed.

**REMOVE**

Remove the system identified by either AFFIELD or AFTOKEN from the specified affinity field. The required parameters are AFFIELD and either ID or AFTOKEN.

**Note:** If both ID= and AFTOKEN= are specified for a request of ADD, REMOVE, TEST, or MOVE, the ID is first converted into a token and placed in the field pointed to by AFTOKEN=. After this conversion, the requested function is performed.

**RETURN**

Build an affinity token from the passed ID field and return it in the AFTOKEN field. The required parameters are ID and AFTOKEN.

**REVERSE**

Remove all systems currently in the affinity field and add all the systems that are not in the field. The required parameter is AFFIELD.

**TEST**

Determine whether the system identified by either ID or TOKEN is represented in the affinity field. The required parameters are AFFIELD, either ID or AFTOKEN, and either PASS or FAIL.

## \$SETIDAW

**Note:** If both ID= and AFTOKEN= are specified for a request of ADD, REMOVE, TEST, or MOVE, the ID is first converted into a token and placed in the field pointed to by AFTOKEN=. After this conversion, the requested function is performed.

### TESTANY

Test the affinity field to determine whether or not the affinity field represents an affinity of ANY. Determine whether the affinity field represents an affinity of ANY. The required parameters are AFFIELD and either PASS or FAIL.

### TESTALL

Determine whether the affinity field represents an affinity of ALL. The required parameters are AFFIELD and either PASS or FAIL. TESTALL is a synonym of TESTANY.

### TESTNONE

Determine whether the affinity field is completely empty. The required parameters are AFFIELD and either PASS or FAIL.

### TESTONLY

Determine whether the system identified by either ID or TOKEN is the only system represented in the affinity field. The required parameters are AFFIELD, either of ID and AFTOKEN, and either of PASS and FAIL.

**Note:** If both ID= and AFTOKEN= are specified for a request of ADD, REMOVE, TEST, or MOVE, the ID is first converted into a token and placed in the field pointed to by AFTOKEN=. After this conversion, the requested function is performed.

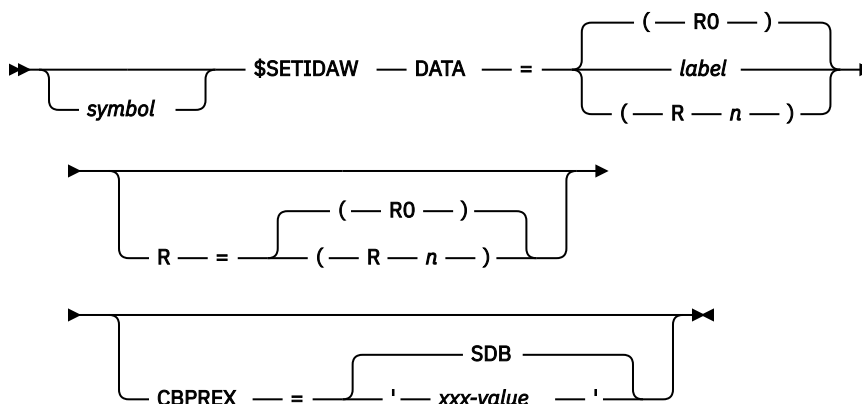
## Return codes

None. For a test request (any of REQUEST=TEST, REQUEST=TESTANY, REQUEST=TESTALL, REQUEST=TESTNONE, REQUEST=TESTONLY), either PASS= or FAIL= (or both) MUST be coded.

## \$SETIDAW – Set indirect data access word (IDAW)

Use \$SETIDAW to store the data buffer address in the xxxIDAWn fields of the specified control block. By default, this macro instruction stores the address passed in register 0 into the xxxIDAW1 field. Use the CBPREX= keyword to specify xxx. \$SETIDAW then sets xxxIDAW2 and xxxIDAW3 fields to the next 2K page boundary addresses when further IDAWs are required.

## Format description



### DATA=

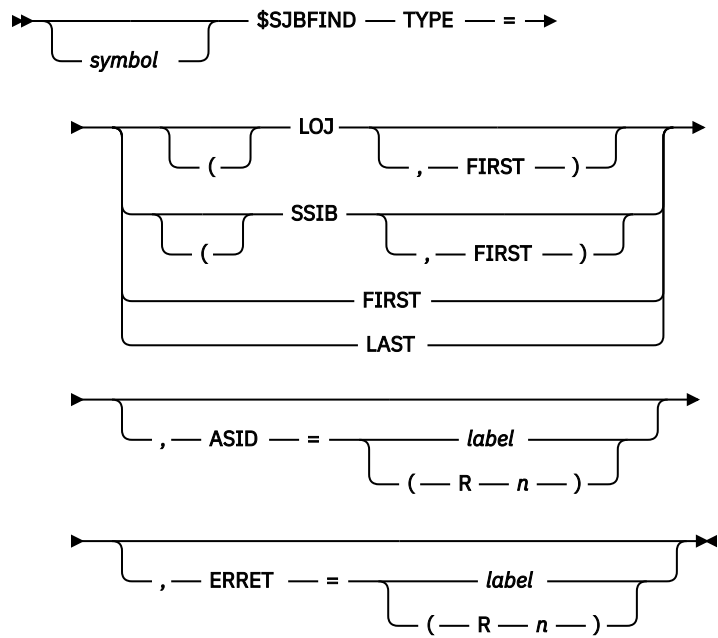
Specifies the label of, or a register containing, the data buffer address to be placed into the xxxIDAWn fields of the specified control block.

### R=

Specifies a work register to be used during \$SETIDAW processing. R=(R0) is the default.



Format description



- TYPE=**  
Specifies the type of SJB to locate, as follows:
- | Type                | Meaning   |
|---------------------|---|
| <b>LOJ</b>          | Life-of-job SJB   |
| <b>(LOJ,FIRST)</b>  | Specifies the life-of-job SSIB SJB and if not there, the first SJB for the address space. |
| <b>SSIB</b>         | Subsystem information SJB   |
| <b>(SSIB,FIRST)</b> | Specifies the caller's SSIB SJB and if not there the first SJB for the address space.     |
| <b>FIRST</b>        | First SJB of the address space  |
| <b>LAST</b>         | Last SJB of the address space   |
- ASID=**  
Specifies the address space identifier (ASID) that is to be used if TYPE=FIRST or TYPE=LAST is also specified. If label is specified, it must indicate a halfword area. The default is the current ASID.
- ERRET=**  
The address to branch to if an error occurs.

Return codes

The following return codes (in decimal) are returned in register 15.

- | Return Code | Meaning   |
|-------------|-----------|
| 0           | SJB found |

- 4 SJB not found
- 8 SJB not found because HASB is missing.
- 12 SJB not found because subsystem names are different.

## Programming requirement

Be certain to include \$TRE on the \$MODULE call. The \$SJBFIND calls \$GETHP which requires this mapping.

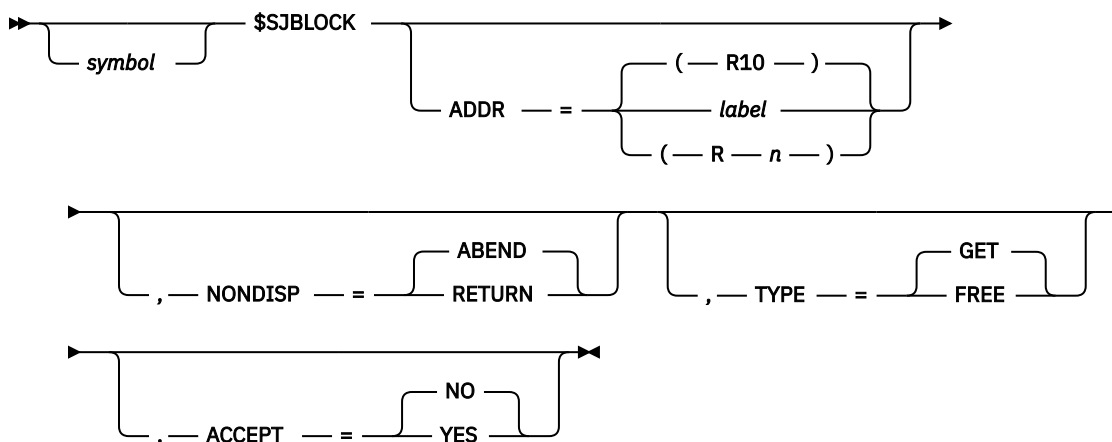
## Environment

- User environment.
- MVS WAIT cannot occur.

## \$SJBLOCK – Lock a specific subsystem job block (SJB)

Use \$SJBLOCK to lock or release a specific subsystem job block (SJB).

## Format description



### ADDR=

Specifies the label or a register that contains the address of the SJB to be locked. ADDR=(R10) is the default.

### NONDISP=

Specifies that a TYPE=GET requester wants control to be returned (RETURN) or abnormally terminated (ABEND) if the SJB lock is held by a non-dispatchable task.

### TYPE=

Specifies whether to obtain (GET) or free (FREE) the SJB lock. TYPE=GET is the default.

### ACCEPT=

Specifies whether the caller will accept the SJB lock if it is already held by the same TCB. The default is NO.

## Return codes

The following return codes (in decimal) are returned in register 15.

Return Code	Meaning
-------------	---------

## \$SJBRO

- 0**  
Successful completion, SJB lock obtained or freed
- 4**  
SJB lock already held by the caller
- 8**  
Disastrous error, specified SJB is terminating, lock is not obtained.
- 12**  
SJB lock owner is currently non-dispatchable, lock is not obtained.

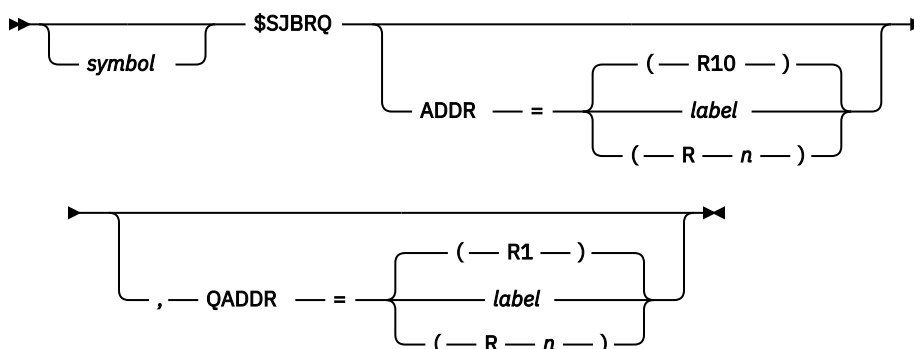
## Environment

- User environment.
- MVS WAIT cannot occur.

## \$SJBRO – Requeue a specific subsystem job block (SJB)

Use \$SJBRO to call the \$SJBRO service routine to requeue the SJB to the \$SVJ queue specified by the calling routine.

## Format description



### ADDR=

Specifies the label or a register that contains the address of the SJB to be requeued. ADDR=(R10) is the default.

### QADDR=

Specifies the label or a register that contains the address of the new \$SVJ queue to receive the SJB.

## Return codes

The following return codes (in decimal) are returned in register 15.

### Return Code

#### Meaning

- 0**  
Successful completion, SJB requeued.
- 4**  
SJB not requeued, no queue header available.

## Environment

- User environment.
- MVS WAIT cannot occur.

# \$SSIBEGN – Begin a subsystem interface (SSI) function

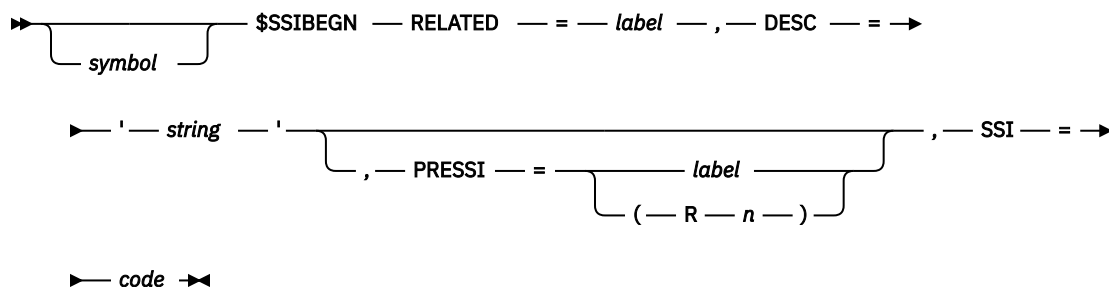
Use \$SSIBEGN to define the beginning of a subsystem interface (SSI) function. This macro instruction calls all necessary routines needed to initialize the subsystem interface's function.

This macro calls the PRESSI routine to determine if this SSI request needs to be processed. This determination prevents unnecessary processing, such as obtaining storage when it is not required. This macro is particularly useful to prevent performance degradation during MVS broadcast calls. PRESSI determines whether the calls need to be serviced by the subsystem; if it does not, control is immediately returned to the caller. Therefore, this macro will allow the SSI to bypass unnecessary recovery and initialization processing.

The PRESSI routine runs in an extremely limited environment; note the following restrictions:

- Save area services are not available
- Alteration of registers R11, R13, and R14 is not allowed
- There is no recovery environment
- PRESSI runs under the key and authority of the caller.

## Format description



### RELATED=

Specifies the label of the \$SSIEND macro instruction which ends the SSI function which this \$SSIBEGN starts.

### DESC=

Specifies a character string that describes the SSI function you are defining. You can specify up to 38 characters and must enclose the string within single quotation marks. You can use the \$D SSI command or D SSI display-only initialization statement to display this string.

### PRESSI=

Specifies the label or a register that contains the address of a routine that receives control to determine if this SSI request should be processed by the subsystem interface (SSI).

### SSI=

Specifies the SSI number, between 1 and 256, for the SSI function you are defining. You can specify either a number or a symbol.

The following return codes (in decimal) are returned in Register 15.

### Return Code On exit

#### Meaning

#### <0

Processing successful. \$SSIBEGN macro expansion should acquire the necessary control blocks and invoke the SSI function routine.

#### >=0

Processing failed. Return immediately to the SSI caller.

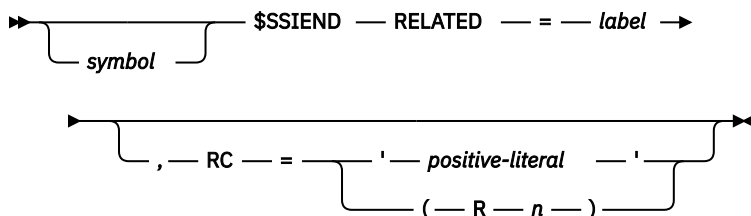
## Environment

- User environment.
- MVS WAIT cannot occur.

## \$SSIEND – End a subsystem interface (SSI) function

Use \$SSIEND to call the \$SSIEND service routine to define the end of the SSI function. Before returning to the caller, the caller's registers are restored.

### Format description



#### **RELATED=**

Specifies the label of the \$SSIBEGN macro instruction which starts the SSI function which this \$SSIEND ends.

#### **RC=**

Specifies return code or a register that contains the return code that is to be returned to the caller.

The following return codes (in decimal) are returned in Register 15.

#### **Return Code Before exit**

##### **Meaning**

#### **>=0**

The actual return code is placed in the SSOBRETN field and register 15 will be reset to zero on return to the caller.

#### **<=0**

Return immediately to the caller and place the absolute value of the return code in register 15.

#### **-8**

Subsystem exists, but is not active.

#### **-12**

Subsystem does not exist.

#### **-16**

Function not completed, disastrous error.

#### **-20**

Logical error.

## Environment

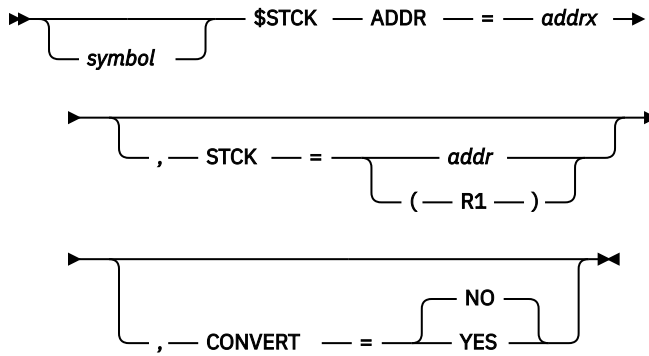
- User environment.
- MVS WAIT cannot occur.

## \$STCK – Call the \$STCK service routine

Use the \$STCK macro instruction to call the \$STCK (store clock) service routine. This service routine gets the time from the TOD clock and stores it at the location you specify.



## Format description



### ADDR=

Specifies the address of an 8-byte storage area or register whose content is the address of an 8-byte storage area. R0 is loaded with this address and the service routine is called if the STCK instruction could not get the TOD clock setting. The field contains, on exit, the time in either STCK or packed decimal format. This parameter is required.

### STCK=

Specifies the address of the field containing the time to be converted from the TOD clock format to packed decimal. If not specified, a STCK instruction will be used to obtain the time.

### CONVERT=

Specifies whether or not the obtained time is to be converted to packed decimal format.

#### YES

The time should be converted to packed decimal format.

#### NO

The time should not be converted to packed decimal format.

## Environment

- JES2 main task.
- \$WAIT cannot occur.

## \$STIMER – Set interval timer

Use \$STIMER to set a time interval for the programmed interval timer.

## Format description



### loc

Specifies the address of a JES2 timer queue element (TQE). Before this macro instruction is executed, the TQE must be initialized. TQETIME must be initialized with the interval to be set in the following manner:

- If x seconds are desired, the set TQETIME to x.
- If y hundredth-seconds (0.01 seconds) are desired, then set TQETIME to the two's complement of y.

TQEPCE must be initialized with the address of the processor control element (PCE) to be posted.

## \$STMTLOG

If register notation is used, the address must be loaded into the designated register before the execution of this macro instruction.

**Note:** An unlimited number of independent \$STIMER time intervals can be active at any time if each has been furnished with a unique JES2 timer queue element.

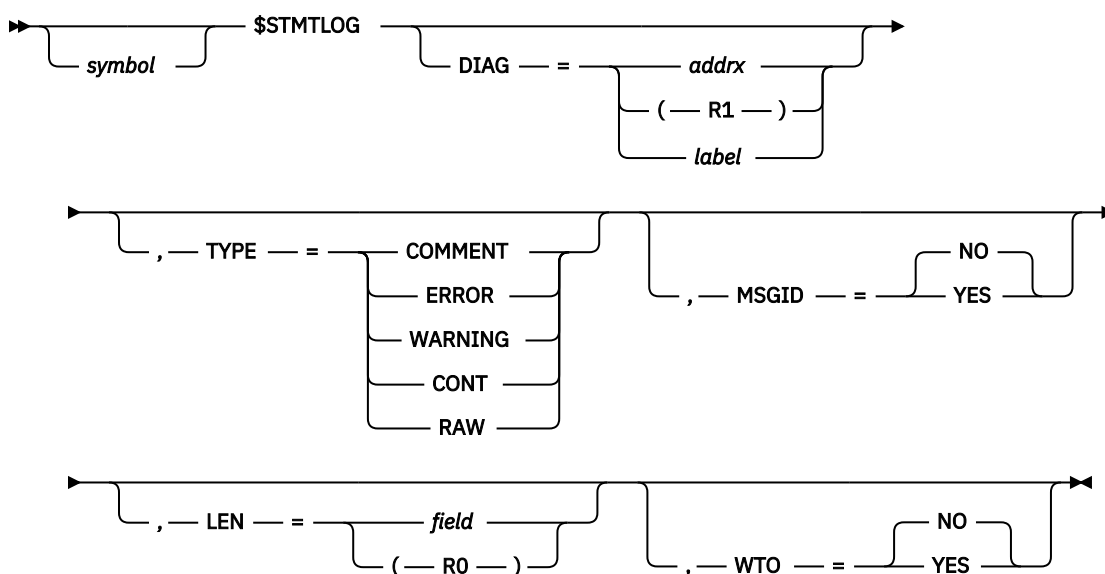
## Environment

- Main task.
- \$WAIT cannot occur.

## \$STMTLOG – Log an initialization statement

Use \$STMTLOG to log initialization statements and related diagnostic information. This macro can be used by either JES2 or Exit 19 (Initialization Statement).

## Format description



### DIAG=

Specifies the address of the diagnostic information associated with the last analyzed initialization statement or specifies the actual text of the diagnostic information when “label” is the keyword value. The message can be formatted to contain a message ID and/or message length as well as the actual text. The following table provides the required format information:

MESSAGE ID	LENGTH	DIAGNOSTIC	FORMAT
NO	NO	Text	‘DDDDDD...’
YES	NO	Text	‘XXXDDD...’
NO	YES	Text	***Error***
YES	YES	Text	***Error***
NO	NO	Address	LDDDDDD...
YES	NO	Address	XXXLDDDD...
NO	YES	Address	DDDDDDDD...
YES	YES	Address	XXXDDDDDD...

MESSAGE ID	LENGTH	DIAGNOSTIC	FORMAT
<p><b>*</b></p> <p>XXX = message ID</p> <p>L = length</p> <p>DDD = diagnostic</p>			

**TYPE=**

Specifies the type of diagnostic message that is to be logged for the last analyzed initialization statement.

**COMMENT**

Log the diagnostic information to hardcopy only.

**WARNING**

Log the diagnostic information to hardcopy if the source of the last analyzed initialization statement is not the console. If the source is a console also log the diagnostic information to the console.

**ERROR**

Log the current parameter statement to the console and hardcopy along with the diagnostic information.

**CONT**

The message is a continuation of a previous (non-raw) message.

**RAW**

Data that is passed is to be written with no prefix being added.

**MSGID=**

Specifies whether (YES) or not (NO) a message ID (\$HASPnnn) is included in the diagnostic text that is passed to the STMTLOG routine.

**YES**

Indicates that the message ID is supplied as part of the diagnostic text.

**NO**

Indicates that the message ID is not supplied as part of the diagnostic text.

**LEN=**

Specifies the address of the length (1-80 characters) of the diagnostic message. If this keyword is not specified, JES2 assumes that the length of the message is imbedded in the message.

**WTO=**

NO|YES - DIAG points to an MF=L WTO that is issued as a WTO if this is normal processing or written to the log if this is the initialization deck checker.

**Note:** When no operands are specified, the last analyzed initialization statement is logged.

## Environment

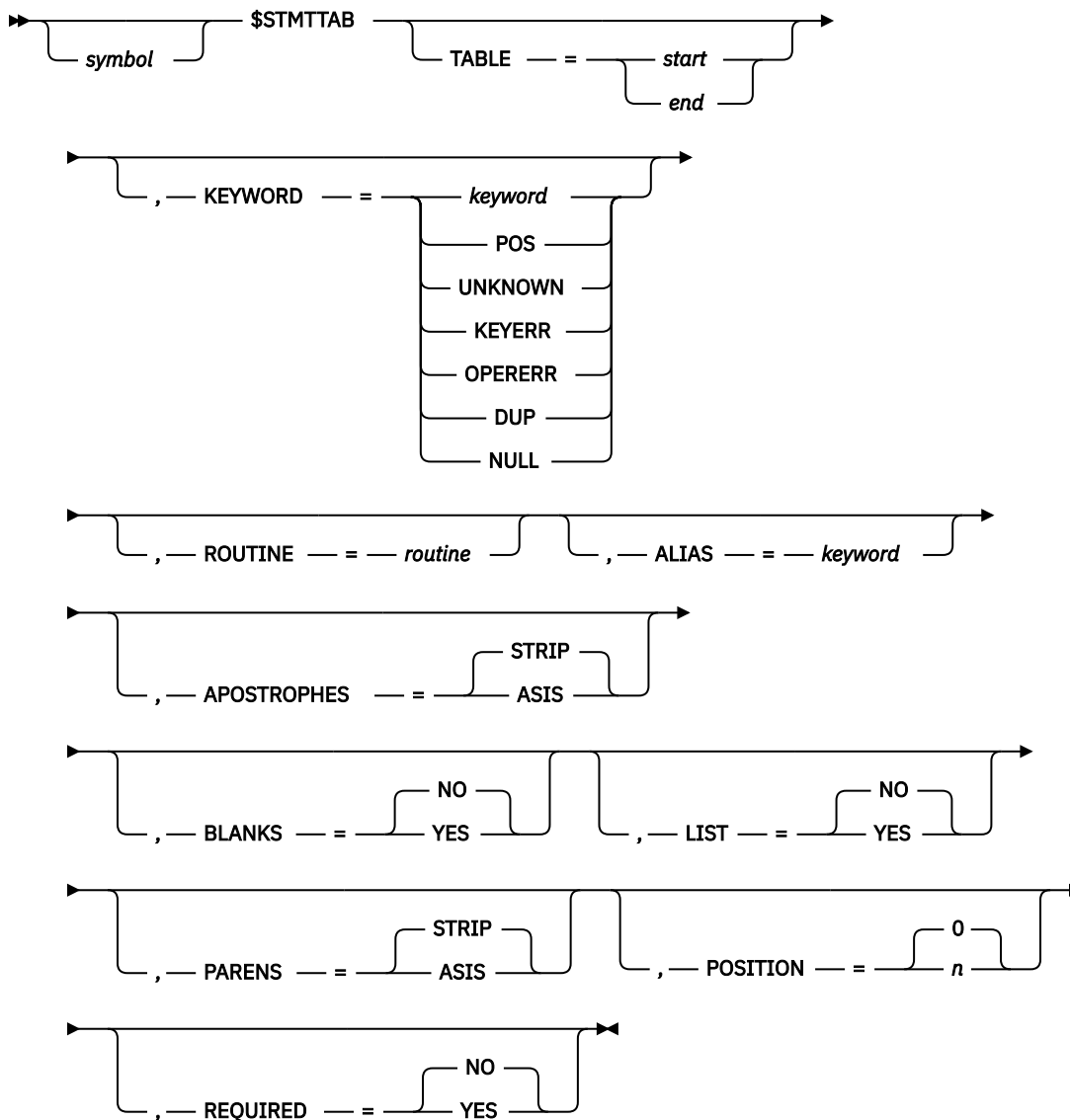
- Only Exit 19 during JES2 initialization.
- \$WAIT cannot occur.

## \$STMTTAB – Card Scan table entry

Defines the table entries used by the RCARDSCN service. Each entry defines either a keyword, positional operand, or condition that can exist in a JCL or JECL statement and a processing routine. Various processing options are also defined in this table. The first entry must be a TABLE=HASP or TABLE=USER entry. The last entry is a TABLE=END.

The label on TABLE=HASP or USER is passed to RCARDSCN in register 0 and the \$JRW (with the statement to be processed) is passed in register 1.

## Format description



### TABLE=

Indicates the start and end of a statement scan table. A start table entry must have a label and specify TABLE=HASP for IBM supplied tables and TABLE=USER for user defined tables. The final table entry specifies TABLE=END. Each table must have a single start and end table entry.

### KEYWORD=

Indicates keyword or positional operand being scanned for or condition to be processed. Enclose keywords in apostrophes. KEYWORD= can also be passed as one of these special values.

### POS

Positional value (no KEYWORD= in statement).

### UNKNOWN

Keyword that does not match any other keywords.

### KEYERR

Keyword error detected (generally length error).

**OPERERR**

Operand error detected (generally length error).

**DUP**

Keyword that exists more than once in the card.

**NULL**

There are no operands to process.

**ROUTINE=**

Indicates routine that is given control to when the keyword or condition specified on KEYWORD= occurs. If ROUTINE= is not specified, the keyword or condition is ignored (that is, processed as if the routine is called and returned a zero return code).

**ALIAS=**

Indicates that an alias is to be used for the keyword enclosed in apostrophes. ALIAS is only valid if the value specified for KEYWORD= is enclosed in apostrophes

**APOSTROPHES=**

Remove all apostrophes before calling routine (STRIP) or leave apostrophes alone (ASIS). STRIP is the default.

**BLANKS**

Skip blanks after keyword (YES) looking for value (no '=' required) or treat blanks as the end of statement (NO). NO is the default.

**LIST**

Process value as a list or not. LIST=NO indicated this is not a list (the default). LIST=(YES,size) says value is a list and passed to the processing routine as an array of elements. Each element has the size specified.

**PARENS**

Remove enclosing parenthesis before calling routine (STRIP) or leave parentheses alone (ASIS). STRIP is the default.

**POSITION**

Indicates which positional operand to process with this table entry. 0 (default) is all positionals. Otherwise specify n from 1 to 255.

**REQUIRED**

Keyword is required (YES) or optional (NO). If a required keyword is not specified, then the processing routine gets control with -1 as the length of the value (JRWRSCNL). NO is the default.

The following are the register values on entry to the specified routine:

**R0:**

N/A

**R1:**

JCT base address

**R2:**

N/A

**R3:**

N/A

**R4:**

N/A

**R5:**

N/A

**R6:**

N/A

**R7:**

N/A

**R8:**

N/A

**R9:**

N/A

**R10:**

JRW address

**R11:**

HCCT address

**R12:**

N/A

**R13:**

Save area address

**R14:**

Return address

**R15:**

Entry address

The following are the register values and conditions on exit from the specified routine:

**R0:**

N/A

**R1:**

80 byte message to add to JCL stream or zero

**R2:**

Unchanged

**R3:**

Unchanged

**R4:**

Unchanged

**R5:**

Unchanged

**R6:**

Unchanged

**R7:**

Unchanged

**R8:**

Unchanged

**R9:**

Unchanged

**R10:**

Unchanged

**R11:**

Unchanged

**R12:**

Unchanged

**R13:**

Unchanged

**R14:**

N/A

**R15:**

Return Code

## Return codes

The following return codes (in decimal) are returned in register 15.

### Return Code

#### Meaning

0

Processing successful.

4

Error encountered, fail job at end of input (keep scanning statement).

8

Error encountered, stop scanning statement and kill job.

## \$STORE – Store registers in the current processor save area

Use \$STORE to store one or more registers in the current processor save area (that is, the one associated with the most recently issued \$SAVE macro instruction). The stored registers are returned to a calling routine on execution of a \$RESTORE macro instruction.

## Format description

➤ symbol \$STORE — *list* — REGS= — AREGS= — QREGS= ➤

### list

Specifies a list of one or more registers, and/or groups of registers to be stored. If more than one register is to be stored, the entire list must be enclosed in parentheses.

A register group is indicated by a pair of registers enclosed in parentheses. All registers, beginning with the first register specified and ending with the second register, are stored. The order of storing a group of registers is: R14, R15, R0-R12. If the list consists of a single group, the outer (list) parentheses are not required.

**Note:** All registers must be specified symbolically. The accepted register symbols are: R0, R1, R2, . . . , R15.

Examples:

```
Store register 2
$STORE (R2) or
$STORE R2
Store registers 15 through 8
$STORE ((R15,R8)) or
$STORE (R15,R8)
Store register 3 and register 10
$STORE ((R3), (R10)) or
$STORE ((R3),R10) or
$STORE (R3,(R10))
Store registers 0, 3 through 5, and 8
$STORE (R8, R0,(R3, R5))
```

**Note:** The sublist order is unimportant.

### REGS=

A list of general purpose registers to be stored.

### AREGS=

A list of access registers to be stored. This parameter is not valid in the USER, FSS, or SUBTASK environment.

### QREGS=

A list of registers for which both the access register and the general purpose register are to be stored. Access registers are never stored in the USER, SUBTASK, or FSS environments.

## Environment

- All environments.
- \$WAIT cannot occur.

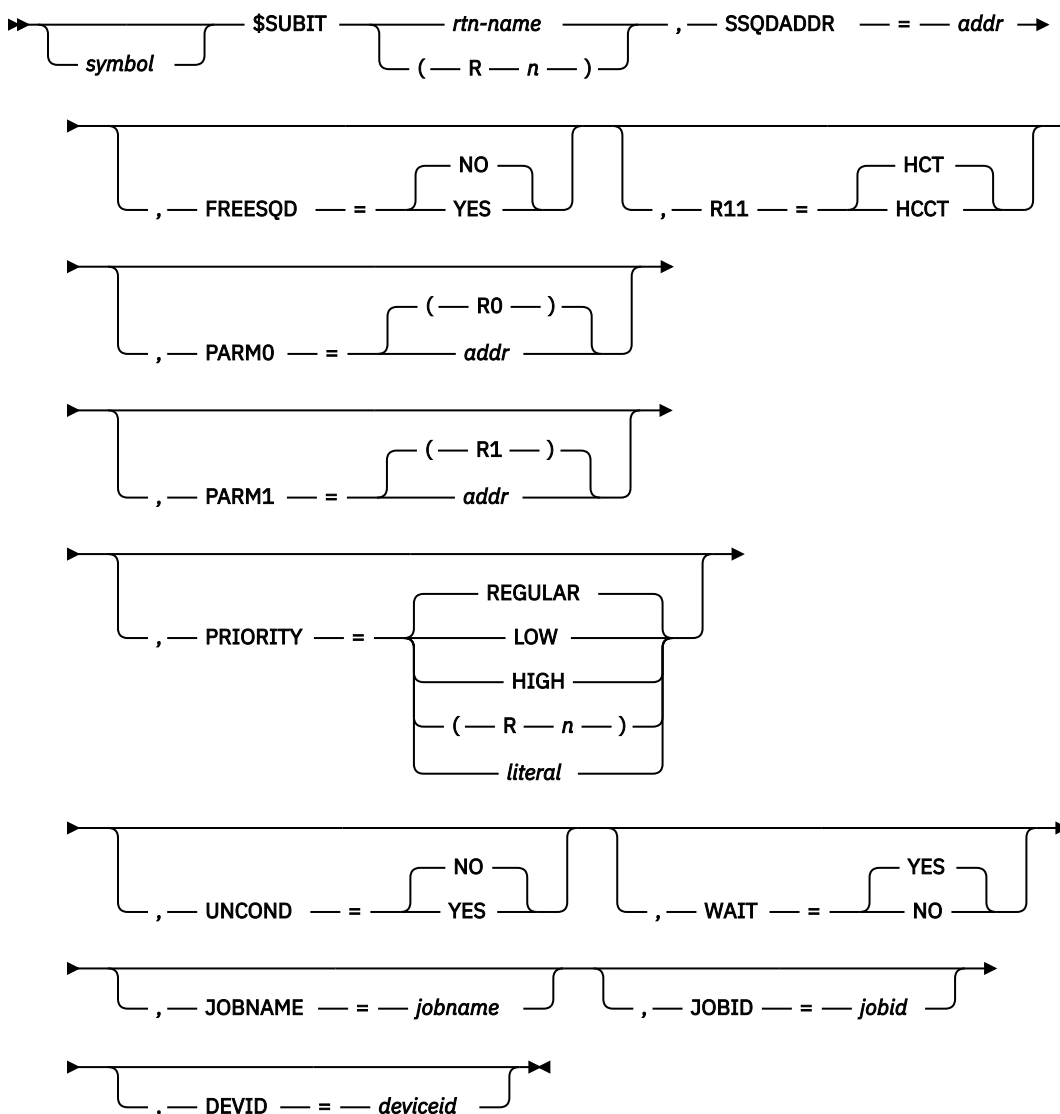
## \$SUBIT – Initiate subtask queueing

Use \$SUBIT to invoke a service routine under a subtask. Use \$SUBIT to provide services, from the main task, that might:

- Cause an MVS WAIT to occur.
- Perform input or output operations.
- Perform intensive calculations.

\$SUBIT places the request for processing by a specified routine on a subtask work queue.

## Format description



### rtn-name|Rn

Specifies the name (1 to 8 characters), or a register that contains the address, of the routine that is to run under the subtask. This is a required, positional parameter.



**SQDADDR=addr**

Specifies the address of the subtask queue descriptor (\$SQD). This parameter is required.

**Note:** The \$SQD must contain any parameters, or the address of any parameter lists, the routine requires.

**FREESQD= YES|NO**

Specifies whether (YES) or not (NO) a \$SUBIT call obtains and releases an SQD (subtask queue descriptor). FREESQD=YES is mutually exclusive with SQDADDR= and WAIT=YES.

**R11=HCT|HCCT**

Specifies the communication table address the routine called by the subtask is to use.

**Default:** HCT

**PARMO=addr|R0**

This is the value in register 0 passed to the routine by the subtask. The value of this parameter depends on the routine invoked. If you specify R0, or allow this to default, you must place whatever the routine requires in R0 before invoking \$SUBIT.

**Default:** R0

**PARM1=addr|R1**

This is the value in register 1 passed to the routine by the subtask. The routine uses this as the address of its parameter list. If you specify R1, or allow this to default, you must place whatever the routine requires in R1 before invoking \$SUBIT.

**Default:** R1

**PRIORITY=LOW|REGULAR|HIGH|(Rn)|literal**

Specifies the priority of this request. If specified as a register (Rn), the low-order byte of the register must contain the value associated with the priority required.

If specified as a literal, the literal must be equated to one of the values associated with the priority required. The values and the associated priority are:

**Value Passed****Priority Required**

**0**

Regular

**1**

High

**2**

Regular

**other**

Low

**Default:** Regular

**UNCOND=YES|NO**

Specifies whether the JES2 main task will issue this request when no general purpose subtasks are attached. (An MVS WAIT may occur.)

**Default:** NO

**Note:** Only use UNCOND=YES when JES2 must perform the function you are requesting under any condition even at the expense of performance. **Severe performance degradation will occur if JES2 did not attach any general purpose subtasks.**

**WAIT=YES|NO**

Specifies whether \$SUBIT is to return after queueing the request to the subtask. If you specify WAIT=NO, code a \$WAIT for the \$SQD to post the ECB identified by field SQDXECB in \$SQD. WAIT=YES is mutually exclusive with FREESQD=YES.

**Default:** YES

**Note:** If FREESQD=YES is specified, WAIT= has no default because the ECB, which is part of the SQD, is being freed by the subtask.

**JOBNAME=**

Specifies an 8-byte field containing the job name of the job associated with the request. In the JES2 main task, this defaults to the job pointed to by PCEJQE.

**JOBID=**

Specifies an 8-byte field containing the job id of the job associated with the request. In the JES2 main task, this defaults to the job pointed to by PCEJQE.

**DEVID=**

Specifies a 3-byte field containing the device id of the device associated with the request. In the JES2 main task, this defaults to the device pointed to by PCEDCT.

## Return codes

The following return codes (in decimal) are returned in register 15.

**Return Code****Meaning****0**

Processing successful (no errors).

**4**

Invalid routine address. \$SUBIT did not call the routine. Do not resubmit the request.

**8**

An abend occurred in the called routine. Do not resubmit the request.

**12**

A subtask error occurred before calling the routine. Retry the request.

**16**

A subtask error occurred after calling the routine. Retry the request.

**20**

Processing failed. JES2 did not attach any general purpose subtasks during initialization. You receive this code only if you coded UNCOND=NO.

**24**

Processing failed. An abend occurred in \$SUBIT.

**28**

Processing failed. A valid \$SQD was not available.

## Environment

- JES2 main task.
- \$WAIT can occur.
- MVS WAIT can occur if you specify UNCOND=YES.
- Allowed in USER environment from NETSRV address space.

## \$SYMREC – Create and issue a symptom record

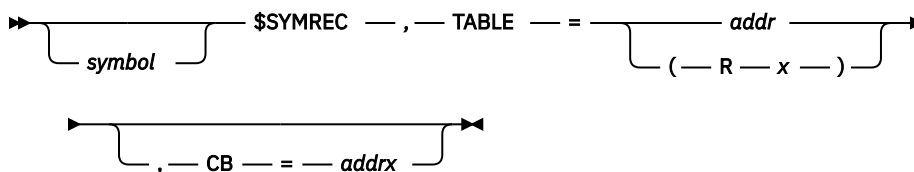
---

Use \$SYMREC to create a symptom record and record the symptom record in the logrec data set or a job log. The data in the symptom record is a description of a programming failure and a description of the environment in which the failure occurred. When JES2 detects an error during execution, it stores diagnostic information into the symptom record and issues the MVS SYMREC macro to log the information.

The symptom record built by \$SYMREC contains five sections. For a detailed description of the type of information in each of the sections, see [z/OS MVS Programming: Assembler Services Guide](#). The \$SYMREC

service fills in the information for sections 1, 2, and 2.1. \$SYMREC also updates section 3 with the component id. Use the \$SYMTAB macro to fill in sections 3, 4, and 5 of the symptom record.

## Format description



### TABLE=

Specifies the address of the beginning of a symptom table to be used to create the symptom record. The symptom table must have been created with a \$SYMTAB TYPE=START macro.

### CB=

Specifies the address of a control block that contains information to fill in the symptom record. If a control block is not specified, information is obtained from the HCT or the HCCT.

## Return codes

The following return codes (in decimal) are returned in register 15.

### Return Code Meaning

- |          |  |
|----------|--|
| <b>0</b> | The symptom record was successfully written (no errors).                         |
| <b>4</b> | Unable to write the symptom record because of a failure from MVS SYMREC service. |
| <b>8</b> | Unable to write the symptom record because storage was not obtained.             |

## Environment

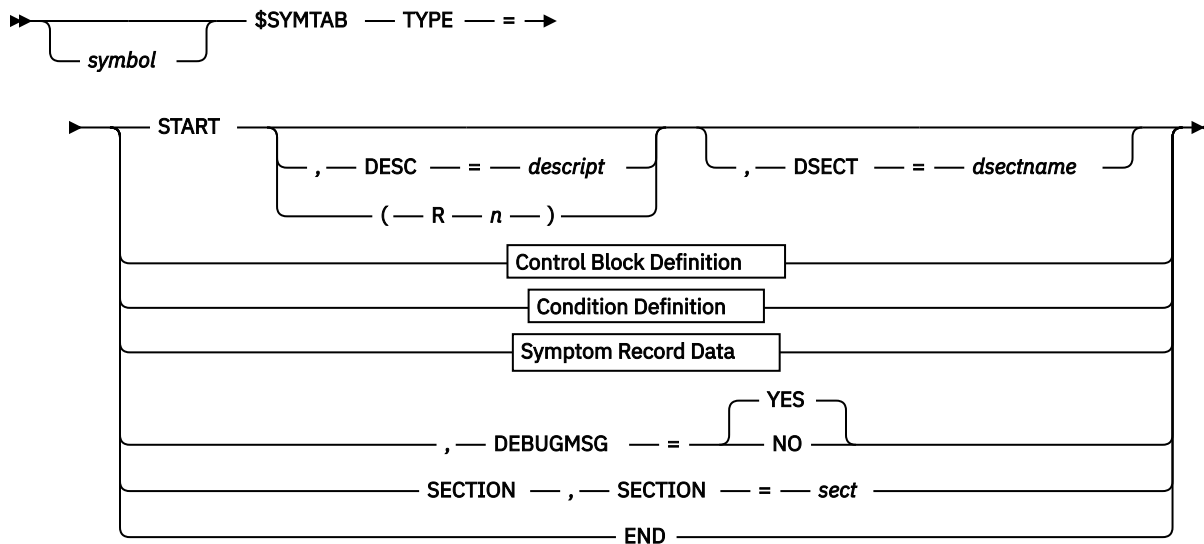
- JES2 main task, JES2 subtask, FSS, and user.
- \$WAIT cannot occur.

## \$SYMTAB – Create a symptom record table

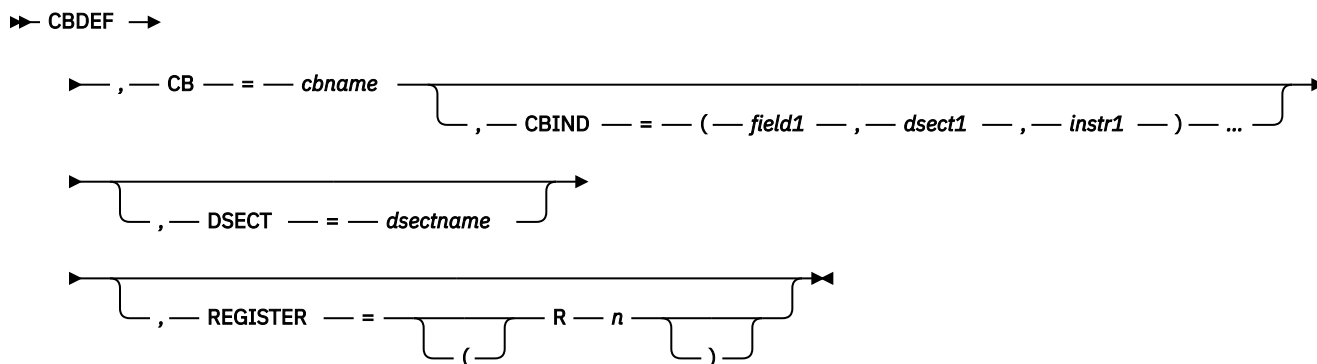
Use \$SYMTAB to create a symptom record table consisting of entries describing information used to fill in sections 3, 4, and 5 of a symptom record created with a \$SYMREC macro. Issue the \$SYMTAB macro multiple times to provide information used to fill in each symptom record. \$SYMTAB entries are created for the following reasons:

- To define the beginning of a symptom record table (TYPE=START). You must create a TYPE=START entry for each symptom record prior to issuing other \$SYMTABs for that symptom record table.
- To define a control block to be used in filling in the symptom record (TYPE=CBDEF).
- To define a condition that will be tested (TYPE=COND).
- To specify data to be placed into a symptom record (TYPE=DATA).
- To specify the beginning of a new section (TYPE=SECTION).
- To define the end of a symptom record table (TYPE=END). You must create a TYPE=END entry for each symptom record after you have defined all the information required in the symptom table with \$SYMTABs.

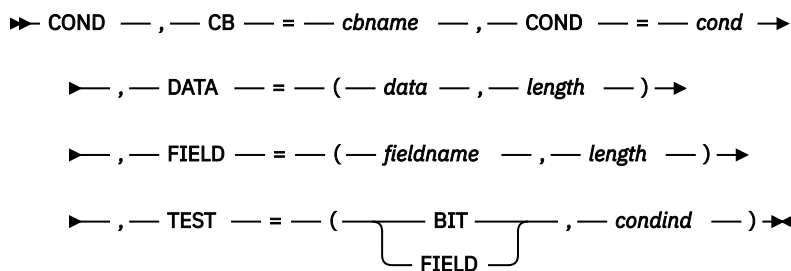
## Format description



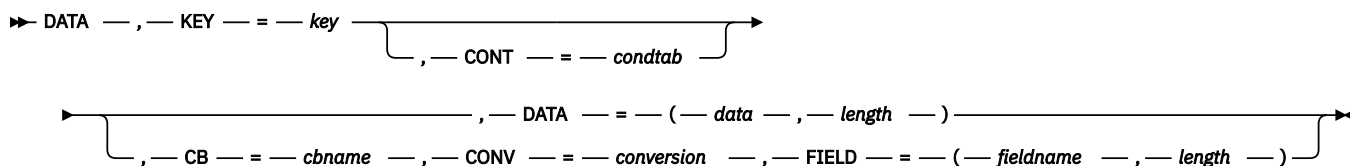
### Control Block Definition



### Condition Definition



### Symptom Record Data



#### TYPE=

Identifies the type of table entry. This parameter is required. Depending upon which TYPE is specified, one or more of the subparameters may be required or optional. Valid types are:

#### START

Specifies the beginning of a table.

**CBDEF**

Defines a control block to be used in filling in the symptom record.

**COND**

Defines conditions to be tested in later \$SYMTAB macros.

**DATA**

Specifies the data to be placed into a symptom record.

**SECTION**

Identifies the start of a new section.

**END**

Specifies the end of a table.

***Depending on which TYPE was coded, the following subparameters may be required or optional:***

**CB=**

Specifies the EBCDIC name of a control block. The meaning of CB= will be different depending on what was specified for the TYPE= keyword.

- If TYPE=CBDEF was specified, CB= will specify the name of the control block being defined. If the CBIND= parameter is not specified on this macro, this name must be the name of the control block you can obtain with the \$GETADDR macro.
- If TYPE=DATA was specified, the CB= parameter specifies the name of the control block that contains the field being formatted. Control blocks must have been previously defined by a TYPE=CBDEF statement. (BASE refers to the control block passed on the \$SYMREC macro.)
- If TYPE=COND was specified, the CB= parameter specifies the control block that contains the field being tested. Control blocks must have been previously defined by a TYPE=CBDEF statement. HCT, HCCT, and BASE are predefined and can be used. (BASE refers to the control block passed on the \$SYMREC macro.)

**CBIND=**

Specifies a series of instructions to locate a control block using a known control block. This parameter must contain a multiple of 3 values, with each set of 3 positional values defining a field name, a DSECT name, and an instruction. The first DSECT must be a control block name that is already known or can be obtained using the \$GETADDR macro. Starting from this control block, the address of the desired control block is found by performing, in succession, the instructions specified against their associated fields in the control block chain. The Rx instructions permitted are L, LA, A, AH, AL, S, SH, SL, and MH.

CBIND= and REGISTER= are mutually exclusive.

**COND=**

Specifies the name of a condition. The meaning of this parameter will be different depending on what was specified for the TYPE= keyword.

- If TYPE=COND was specified, COND= specifies the name of a condition being defined.
- If TYPE=DATA was specified, COND= specifies the set of previously defined conditions which must be met before this SYMTAB can be used. This can be a single value or a set of values, separated by commas, which all must be met.

**CONV=**

Identifies the conversion to be used to create the field. Valid conversions are:

**CHAR**

Data is EBCDIC and needs no conversion.

**HEX**

Binary data is to be converted to EBCDIC hexadecimal data.

**DEC**

Binary data is to be converted into EBCDIC decimal data.

**NONE**

Data is moved, but is not converted. This is the default.

**DATA=**

Specifies constant data. The meaning of DATA= will be different depending on what was specified for the TYPE= keyword.

- If TYPE=DATA was specified, DATA= specifies constant data to be placed in the symptom record. The length subparameter specifies the length of the data. The DATA= keyword cannot be specified with the FIELD= keyword if TYPE=DATA.
- If TYPE=COND was specified, DATA= specifies constant data to be compared to the field data. The length subparameter specifies the length of the data.

**DEBUGMSG=YES | NO**

Specifies whether (YES) or not (NO) you require JES2 to issue message \$HASP805 *jobname* SYMREC ISSUED FROM *module* WITH A DESCRIPTION OF *desc* whenever JES2 requests that MVS write a symptom record to the logrec data set. Although the message is useful to inform you of errors, some symptom records are informational and you might not want to receive this message whenever JES2 requests a SYMREC, particularly for those symptom records that are not likely to indicate actual errors.

**Note:**

1. This specification can be used to suppress a specification of SYMREC=YES on the DEBUG initialization statement.
2. The symptom record is written to the logrec data set regardless of this specification.

DEBUGMSG= is only valid if TYPE=START is also specified.

**DESC=**

Specifies a 32-byte field of descriptive information placed in the 2.1 section of the symptom record. You can specify DESC= as a quoted string, a character string (quotation marks and spaces are not allowed), or a register (R2-R13). If you use register notation, JES2 assumes that when the \$SYMREC macro is issued, the specified register points to a 32-byte constant that is to be used as the description.

If not specified, the description defaults to "JES2 \$SYMREC SERVICE".

**DSECT=**

Specifies the label on the DSECT statement that maps the control block being defined. If not specified, the value of CB= is used.

**FIELD=**

Specifies the name of a field in the specified control block. The meaning of FIELD= will be different depending on what was specified for the TYPE= keyword.

- If TYPE=DATA was specified, the FIELD= subparameter specifies the name of the field that contains data to be placed into the symptom string. The length subparameter specifies the length of the field. If the length is not specified, the assembler length of the field is used.
- If TYPE=COND is specified, the FIELD= subparameter specifies the name of the field that contains data to be tested. The length subparameter specifies the length of the field. If the length is not specified, the assembler length of the field is used.

**KEY=**

Specifies the key describing the data element. For sections 3 and 4 (as defined on the SECTION= keyword of this macro), a structure data base key is used to identify the data. These keys are defined by MVS (For example, KEY=REGS/). For a description of structured data base keys and their use in symptom strings, see *z/OS Problem Management*. For section 5, a four digit hexadecimal constant must be specified to identify the data (for example, KEY=0A34). Keys for section 5 are defined on a component basis.

**REGISTER=**

Specifies, at the time of \$SYMREC macro invocation, which register (R2-R13) contains the address of the control block that is being defined for a TYPE=CBDEF call.

REGISTER= and CBIND= are mutually exclusive.

**SECTION=**

Identifies the number of the section being started. This number must be either 3, 4, or 5.

**TEST=**

Specifies the type of test to be performed for a \$SYMTAB TYPE=COND. The first subparameter specifies whether bits will be tested (BIT) or a field will be tested (FIELD). The tests are performed with FIELD as the first operand and DATA as the second operand. The second subparameter (*condind*) indicates the condition used to set the condition indicator.

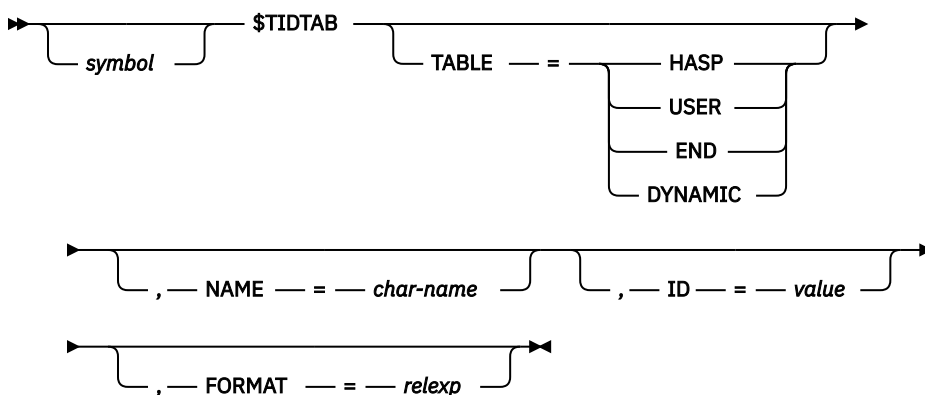
- If BIT was specified, the valid values for *condind* are 'ON', '¬ON', 'OFF', '¬OFF', 'MIXED', and '¬MIXED'.
- If FIELD was specified, the valid values for *condind* are 'EQUAL', '¬EQUAL', 'HIGH', '¬HIGH', 'LOW', and '¬LOW'.

**Environment**

- JES2 main task, JES2 subtask, FSS, and user.

**\$TIDTAB – Generate a trace ID table DSECT**

Use \$TIDTAB to map and generate trace ID (TID) table entries.

**Format description****TABLE=**

Specifies the first entry in the HASP or user TID table or end of a TID table. DYNAMIC specifies that this is a DYNAMIC table. If TABLE= is specified, all other operands are ignored.

**NAME=**

Specifies a 1-8 character name for the TID type. This name is used on the first line of output for the trace record. If NAME= is not specified, DSECT mapping is generated, and all other keywords are ignored.

**ID=**

Specifies a trace identifier, a value from 0-255. The HASP table currently defines identifiers from 0-46.

**Note:** User entries should use identifiers from 255 down to prevent overlap of trace table ID numbers.

**FORMAT=**

Specifies the name of a formatting routine that will format the trace records for this type. If the symbol used as the name of this formatting routine is not defined in the assembly module containing this \$TIDTAB macro, then the TID table field is defined using a VCON rather than an ACON.

**Note:** If you omit all the operands on the \$TIDTAB macro, a DSECT mapping of a TID table entry is generated; otherwise, an actual TID table entry is generated.

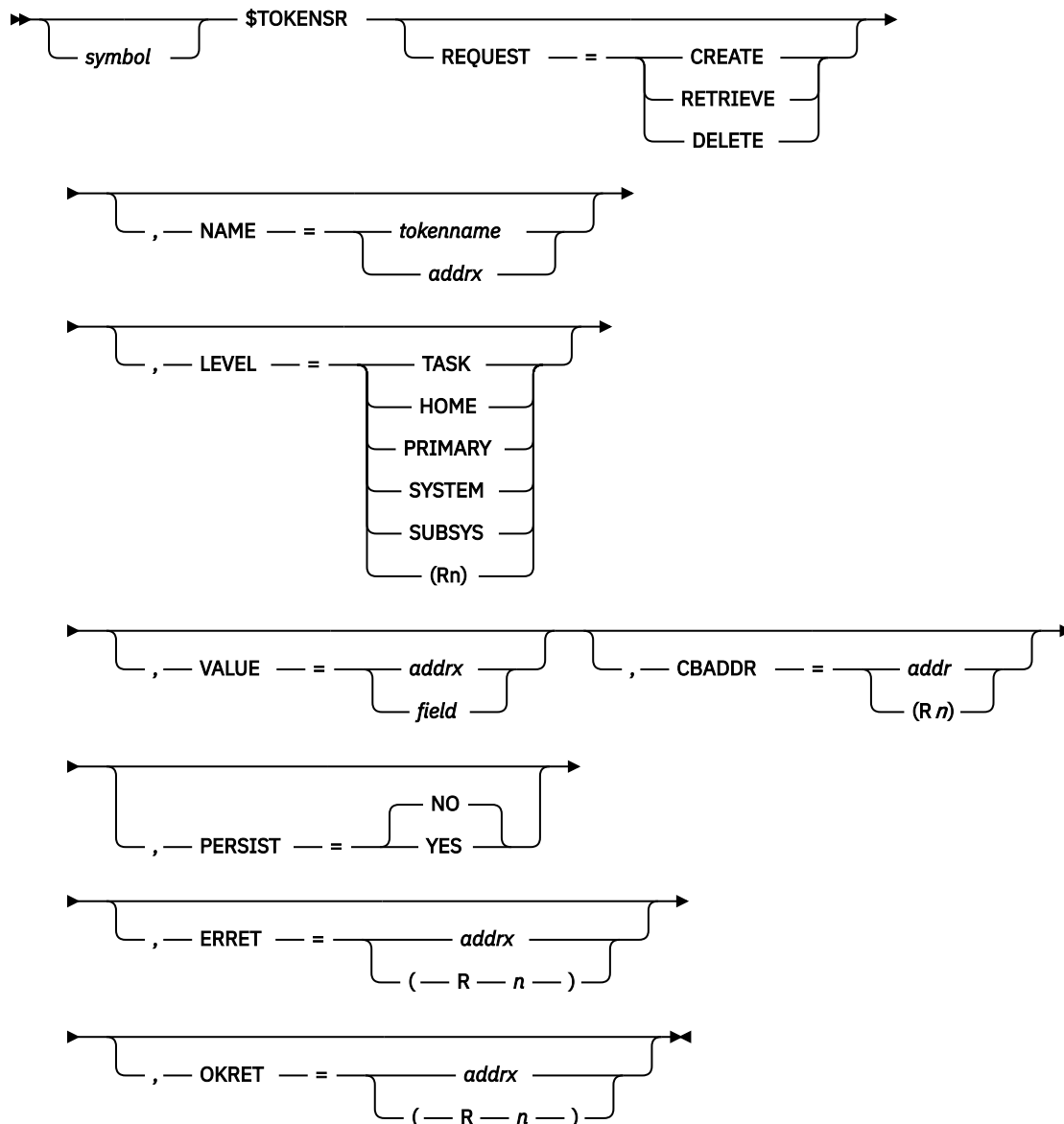
## Environment

- Main task or during JES2 initialization and termination.
- \$WAIT is not applicable – this macro generates a DSECT or a static table entry; it does not generate executable code.

## \$TOKENSR - Create a name/token pair

Creates, retrieves, or deletes a MVS name/token pair.

## Format description



### REQUEST=

Specifies the type of request. Valid values are:

#### CREATE

Creates a name/token pair with specified VALUE= value in the first word of the token.

#### RETRIEVE

Retrieves the first word of the token in R1.



**DELETE**

Deletes the name/token pair.

**NAME=**

Specifies the 1 to 16 bytes of the name of the token, or the address of the field containing the token. For TYPE=SUBSYS, this specifies a 1 to 12-byte name. If the name is enclosed in quotation marks, it is assumed to be the name of the token; otherwise, it is considered to be the name of the field containing the token. For example, consider the following:

- NAME='TOKENNAME' - the actual name of the token is TOKENNAME
- NAME=TOKENNAME - TOKENNAME is a 16-byte field containing the name

**LEVEL=**

Specifies the type of name/token pair to be created or retrieved:

**TASK**

Task level token

**HOME**

Home address space level token

**PRIMARY**

Primary address space level token

**SYSTEM**

SYSTEM level token

**SUBSYS**

SYSTEM level token with the subsystem name automatically included as the last 4 bytes of the 16-byte name

A register containing the value corresponding to the IEANT\_xxx\_LEVEL equate for the specified type (as defined by the IEANTASM macro).

The default is LEVEL=TASK.

**VALUE=**

For TYPE=CREATE, specifies the address of a 16-byte field containing the token data. For TYPE=RETRIEVE, specifies the 16-byte field into which the data should be stored. This parameter is required for TYPE=CREATE. For TYPE=RETRIEVE, if unspecified, the 16 bytes will be returned in R0/R1/AR0/AR1.

**Note:** VALUE= and CBADDR= are mutually exclusive. However, if TYPE=CREATE is specified, either CBADDR= or VALUE= must be specified.

**CBADDR=**

For TYPE=CREATE, specifies the address of a field or control block to place in the first word of the token value. For TYPE=RETRIEVE, specifies a register into which the first word of the token should be placed.

**Note:** CBADDR= and VALUE= are mutually exclusive. However, if TYPE=CREATE is specified, either CBADDR= or VALUE= must be specified.

**PERSIST=**

For TYPE=CREATE, specifies whether the token should remain after the creating address space's job step task terminates. PERSIST=YES is valid for SYSTEM or SUBSYS level tokens only. The default is PERSIST=NO.

**ERRET=**

Specifies the register (2-12) or address of the error routine that is to receive control if the \$TOKENSR request fails.

**OKRET=**

Specifies the address of the routine that is to receive control if the \$TOKENSR request succeeds.

## Return codes

The following return codes (in decimal) are returned in register 15.

Return Code	Meaning
-------------	---------

0	Retrieve successful
4	Retrieve unsuccessful

## **\$TRACE – Trace a JES2 activity**

---

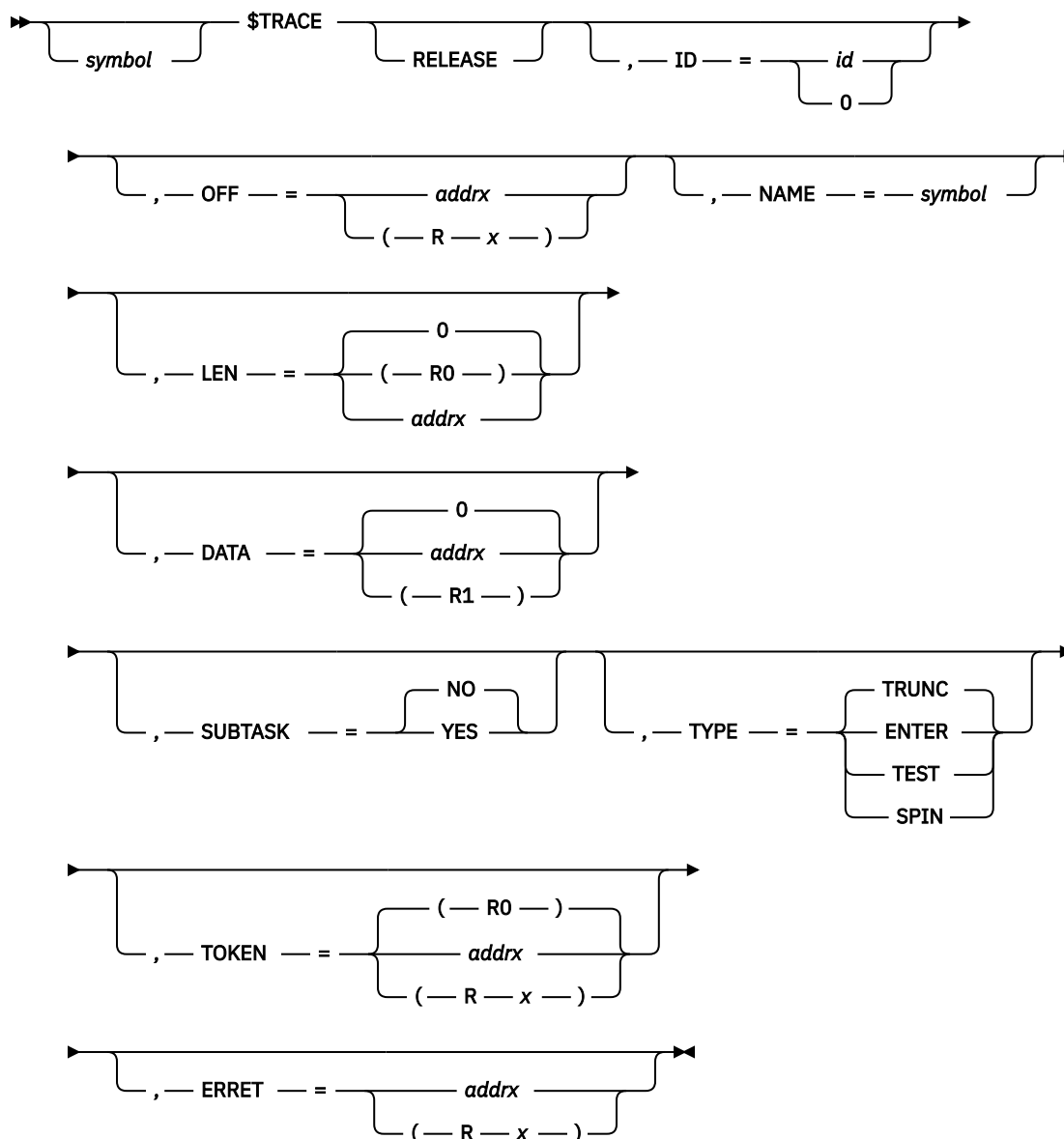
Use the \$TRACE macro instruction to allocate a JES2 trace table entry (TTE) in an active trace table and return its address. Optionally, \$TRACE initializes the TTE based on parameters passed. The JES2 event trace facility is called to perform the TTE allocation.

\$TRACE can be specified anywhere in the JES2 system except in routines running as disabled interrupt exits (for example, an IOS appendage or in cross memory mode as is sometimes the case in the FSS environment). R13 must point to a usable OS-style save area.

On exit, register 1 contains the address of the TTEDATA field in the TTE. Any changes to the TTE must be accomplished before issuing a wait (WAIT or \$WAIT, explicit or implied). A condition code of zero on exit indicates that the TTE was successfully allocated; return code 4 indicates unsuccessful allocation either because tracing is not started or the individual ID is not currently being traced.

In environments other than the JES2 main task, a \$TRACE RELEASE request must be made after the formatted TTE is ready. The \$TRACE facility is serialized using a resource that is obtained by that request.

## Format description



### RELEASE

Specifies the release of the trace buffer. This positional operand must be used in either the user, subtask or the FSS environment after the trace table entry fields are coded.

**Note:** When issuing a \$TRACE RELEASE, TOKEN= must contain the value placed in register 0 when the trace with this ID was originally issued.

### ID=

Specifies the ID associated with this trace entry. The ID is a value between 1 and 255. If 0 is specified, JES2 does not create a TTE but instructs the event trace facility to spin-off the current trace log data set (if logging is active). ID=0 is specified in JES2 routines controlling trace activities and should not be specified outside these areas.

**Note:** IBM trace IDs start from 1 and increase. User trace IDs should start at 255 and decrease to prevent overlap of ID numbers. (See *z/OS JES2 Initialization and Tuning Guide* for the definition of the trace identifiers and the TRACE statements.)

## \$TRACE

### OFF=

Specifies the address that is given control if tracing is not currently being used. If register notation is used, the designated register must be previously loaded with the address.

If this operand is omitted, control is given to the location after the macro expansion with condition code 0.

### NAME=symbol

Specifies the 1- to 8-character identifier to be associated with this macro call. If this operand is omitted, the label symbol used by the \$TRACE macro call is used. If neither is specified, the current CSECT name is used.

### LEN=

Specifies (by a valid expression or through register notation) the length of the trace table entry (TTE) to be allocated. If register notation (R2-R12) is used, the designated register must be previously loaded with the length. The address (addrx) of either a fullword or halfword containing the length can be used if neither register notation (Rx) nor a specified value or equated value is used for length. The maximum length is:

```
(PAGES x 4096) - 68
```

where:

### PAGES

PAGES parameter value on the TRACEDEF initialization statement

### 68

defines the current header lengths in JES2 trace table control block

### DATA=

Specifies the address or a register that points to the location where the data to be logged can be found. If this keyword is specified, all activity for the new TTE is performed by the \$TRACE facility. If this keyword is not specified or DATA=0 is specified and ENVIRON= is specified as USER, SUBTASK, or FSS, you must issue a \$TRACE RELEASE. The returned TTE should be formatted and then a \$TRACE RELEASE must be issued.

### SUBTASK=

Specifies the trace is issued from the JES2 main task or the subtask environment.

### YES

The trace is issued from the subtask environment.

### NO

The trace is issued from the JES2 main task environment.

### TYPE=

Specifies the action to be taken if ID=0.

### ENTER

Generate code to enter the tracing routine without testing the necessity to do so. This assumes that a \$TRACE TYPE=TEST has been used earlier to see if tracing is active and if the trace id is active.

**Note:** Mutually exclusive with OFF=

### TEST

Generates code to test if the trace ID that is specified on the ID= keyword is currently enabled. If this keyword is specified, the OFF= keyword must also be specified. This includes testing if filtering matches for this trace entry. To bypass the filtering tests, specify a second operand of NOFILTER: for example, TYPE=(TEST,NOFILTER). When NOFILTER is specified, the address of the \$TEDE is returned in register 15.

### TRUNC

The current trace table is to be truncated, and the trace table is passed to the trace log processor (if logging is active).

## SPIN

The event trace facility is to spin off the current trace log data set and truncate the current trace table, passing the table to the trace log processor (if logging is active).

**TOKEN**

Specifies the address at which \$TRACE is to place the token returned from the macro service routine or the address of the token \$TRACE is to use when RELEASE is specified.

**addrx**

The address of the storage location of the token.

**Rx**

The register that contains the address of the storage location of the token.

## RO

Register 0 is the default register.

**ERRET=**

Specifies the address to which JES2 gives control if JES2 could not create the current trace id because of a non-zero return code from the \$TRACE service. For example, if the current trace tables are all full.

If you omit this operand, it defaults to the value of the OFF= parameter. For TYPE=ENTER, this operand is required when the DATA= operand is omitted.

**addrx**

The address of the storage location of the token.

**Rx**

The register that contains the address of the storage location of the token.

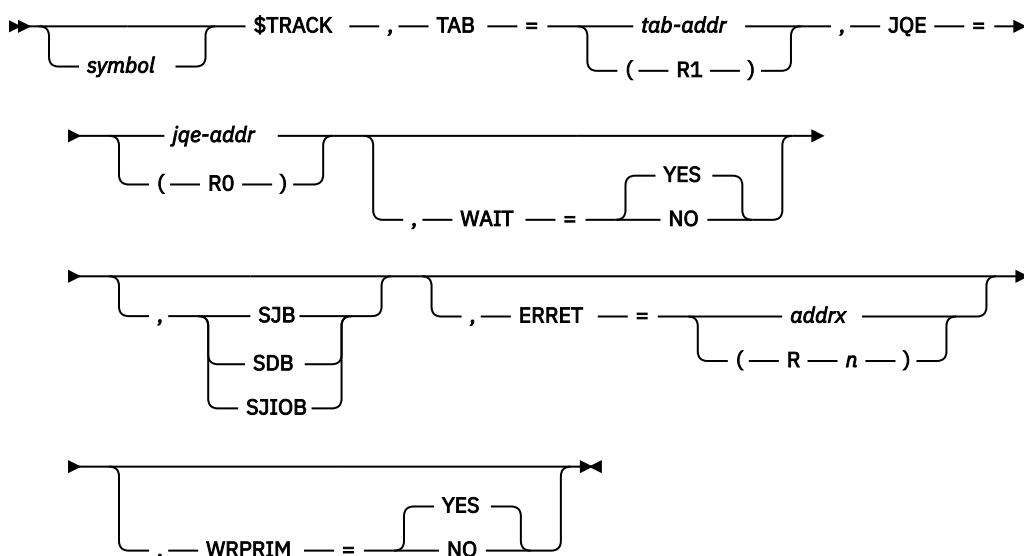
## Environment

- JES2 main task, subtask, user, or functional subsystem (HASPFSM).
- \$WAIT cannot occur; however, in other than the JES2 main task environment, an MVS WAIT can occur.

## \$TRACK – Acquire a direct-access track address

Use \$TRACK to obtain a track address on a JES2 spool volume and return this track address in register 1.

## Format description



**TAB=**

Specifies the address of the track allocation block.

## \$TTIMER

### **JQE=**

Specifies the address of the job queue element (JQE). This is required for the JES2 main task environment.

### **WAIT=**

Specifies whether (YES) or not (NO) to wait if \$TRACK is unable to successfully allocate a track group. If YES is specified, the service routine will issue a \$WAIT TRAK if no tracks are currently available.

### **SJB | SDB | SJIQB**

Specifies the subsystem job block (SJB) address, the subsystem dataset block (SDB) address, or subsystem job input/output block (SIQB) address that JES2 loads into register 10.

#### **Note:**

1. SIQB, SJB, and SDB are mutually exclusive.
2. SJB and SDB are valid only in the user environment.
3. SIQB is valid in the user and subtask environments.

### **ERRET=**

Specifies the address (or register that contains the address) of an error routine that gets control if register 15 contains a non-zero return code from \$STRAK.

### **WRPRIM=**

Specifies whether (YES) or not (NO) to write the primary allocation IOT if a new track group is allocated. WRPRIM=YES is the default.

## Return codes

The following return codes (in decimal) are returned in register 15.

### **Return Code**

#### **Meaning (JES2 Address Space)**

**0**

Allocation successful within the same track group

**4**

Allocation successful in a different track group

**8**

WAIT=NO was specified – no track group returned

### **Return Code**

#### **Meaning (User Address Space)**

**0**

Allocation successful

**8**

Error encountered in \$STRAK

## Environment

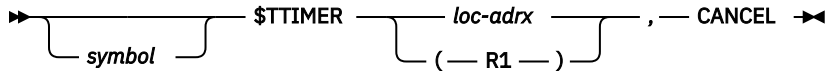
- Main task and user address space.
- \$WAIT can occur.

## \$TTIMER – Test interval timer

---

Use \$TTIMER to obtain the time remaining in the associated time interval that was previously set with \$STIMER macro instruction. The value of the remaining time interval is returned in register 0 in seconds (rounded to the nearest second). The \$TTIMER macro instruction can also be used to cancel the associated time interval.

## Format description

**loc**

Specifies the address of the timer queue element. If register notation is used, the address was loaded into the designated register before the execution of this macro instruction.

If the timer queue element is not active or if the interval expired before the `$TTIMER` macro instruction is executed, the value of the time interval returned is 0.

**CANCEL**

Specifies that the interval in effect should be canceled.

If this operand is omitted, processing continues with the unexpired portion of the interval still in effect.

If the timer queue element is not active or if the interval has expired before the `$TTIMER` macro instruction is executed, the `CANCEL` operand has no effect.

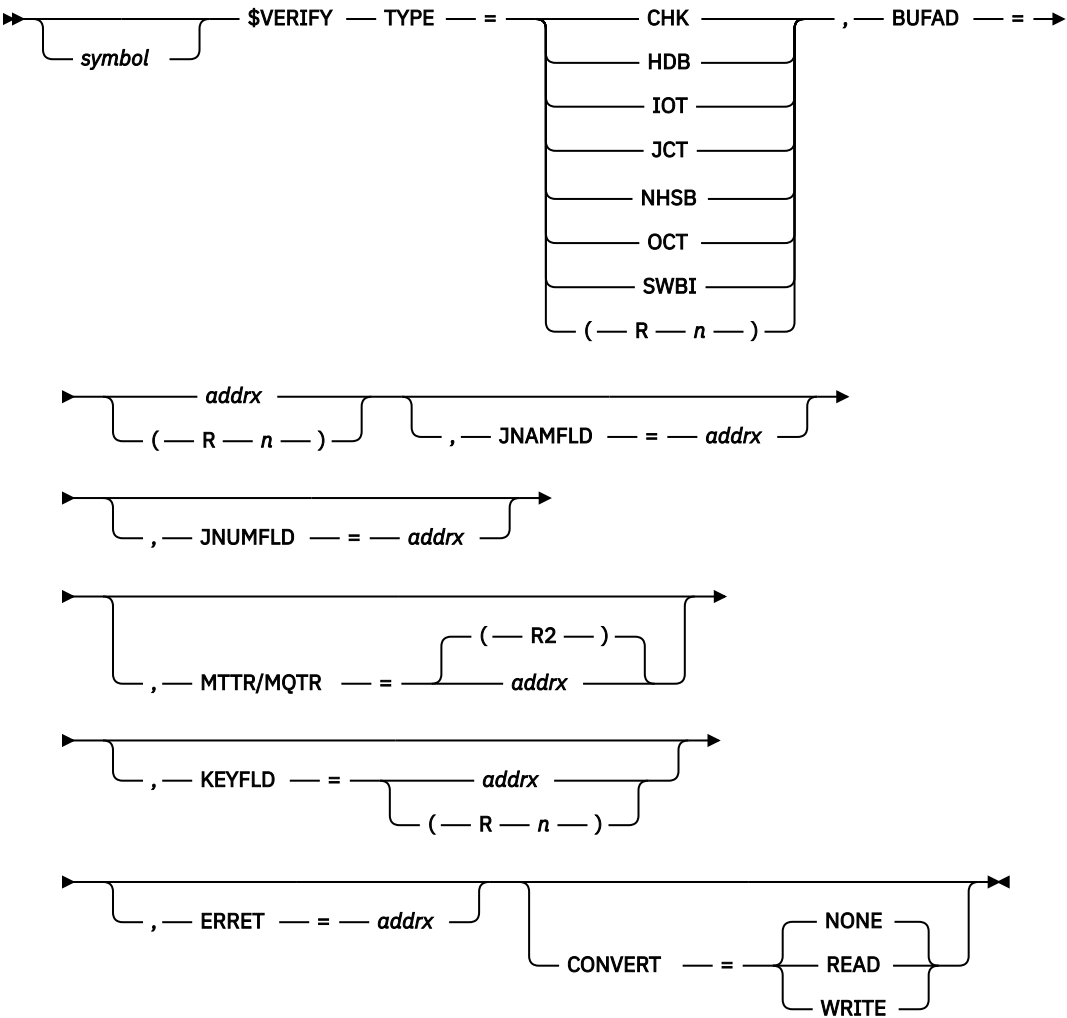
## Environment

- Main task.
- \$WAIT cannot occur.

## \$VERIFY – Verify a control block

Use the \$VERIFY macro instruction to validate control block contents when read in from spool.

Format description



**TYPE=**  
Specifies the EBCDIC control block identifier or a register that contains the address of the 4-byte EBCDIC identifier of the control block to be verified. TYPE= is required for control block verification.

**Control Block**  
**Meaning**

- CHK**  
Checkpoint control block
- DSCA**  
Data set catalog control block
- DSCX**  
Data set index control block
- EMQT**  
Email message queue container
- HDB**  
JES2 SYSIN/SYSOUT data buffer
- IOT**  
input/output table control block
- JCT**  
Job control table control block



**JSMT**

Job Symbol Table container

**NHSB**

NJE network header buffer

**OCT**

Output control table

**SWBI**

Scheduler work block information

**TLBM**

SPOOL track level bit map

**BUFAD=**

Specifies the address of the buffer that contains the control block to be verified.

**ERRET=**

Specifies the address of an error routine that is to get control if a control block error is detected or the control block is not verified.

**JNAMFLD**

Specifies the location of a value to be compared to the content of a control block job name field to determine whether it is a valid control block. This parameter is passed in AR14. This parameter is an optional parameter.

**JNUMFLD**

Specifies the address of a value to be compared to the content of a control block job number field to determine whether it is a valid control block. This parameter is passed in AR15. This parameter is an optional parameter.

**KEYFLD=**

Specifies the address of a field to be used to verify the control block. The value at this address is compared to the control block key field to determine whether the control block is valid. The control block key field for this type is specified by the KEYOFF= value of the verification table you built by using \$VERTAB.

**MTTR= / MQTR=**

Specifies a track address (MTTR or MQTR) that is to be compared to the contents of an MTTR field in the control block to determine whether the control block is valid.

MTTR= and MQTR= are mutually exclusive.

If this parameter is specified, a register or the name of a field is passed to the service routine in R1 (MTTR as 0000 MTTR and MQTR as FOMT TTTR). If MTTR or MQTR is not specified, zero is loaded and passed to the service routine in R1.

**CONVERT=**

Determines any conversion that is performed on the control block. Conversion ensures that the in storage copy of spool control blocks is always the most recent level (for this member) and that versions that are written to spool are compatible with the lowest supported member of the MAS.

**NONE**

Conversion is bypassed, even if a conversion routine exists.

**READ**

The operation that is associated with this \$VERIFY is a READ. If a conversion routine exists for this control block, it is converted (in memory) to the most recent version.

**WRITE**

The operation that is associated with this \$VERIFY is a WRITE.

Ensures that before writing the control block to spool, it is converted to the highest version all members that are supported in the MAS support.

If the MAS is at the most recent level, then no conversion is needed. Otherwise, if a conversion routine exists for this control block, it is converted to the highest version that is supported by all MAS members.

## Return codes

The following return codes (in decimal) are returned in register 15.

### Return Code

## Meaning

0

Control block valid.

4

Control block cannot be verified.

8

Control block invalid.

## Environment

- JES2 main task, user task (HASPFSM address space), and HASPFSM.
- No waits can occur.

**Note:** This macro requires registers 0, 1, 14, and 15.

## \$VERTAB – Build in-line verification tables

Use the \$VERTAB macro instruction to start, to build, and to end the inline verification tables to be used by the \$VERIFY service routine.

Use \$VERTAB to map and generate VER table entries.

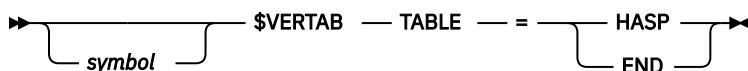
\$VERTAB entries are used to define the start of a user table (\$VERTAB TABLE=USER...) or a JES2 table (\$VERTAB TABLE=HASP...), the end of a table (\$VERTAB TABLE=END) or an entry in a table (\$VERTAB NAME=HUTCH...).

**Note:** The format description that follows breaks the macro into a **boundary form** (the form that starts or ends a table) and an **entry form** (the form that defines each table entry).

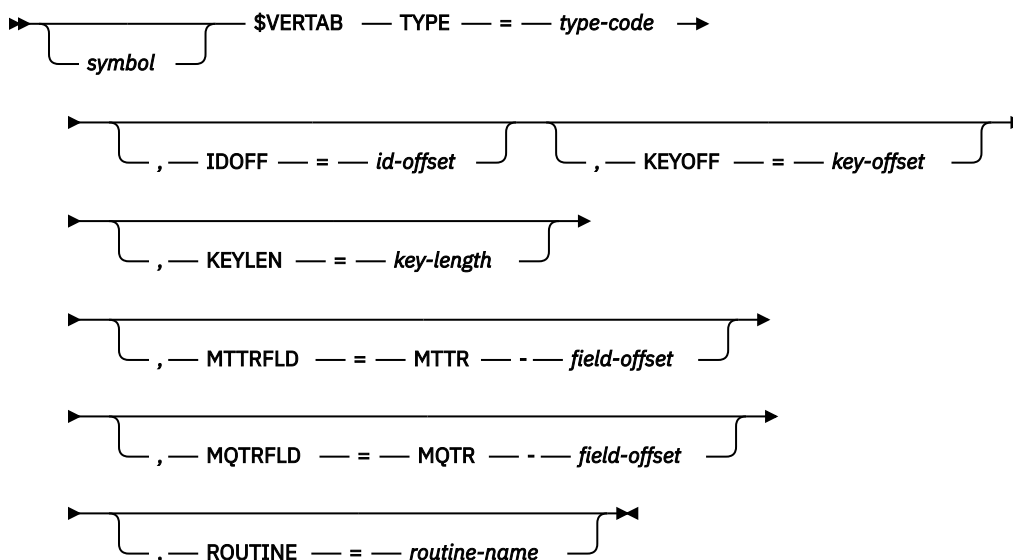
## Format description

The following format descriptions apply:

## Boundary form



## Entry form



### TYPE=

Specifies the EBCDIC id to be used to verify the control block. These types are the same as the types on the \$VERIFY macro.

### IDOFF=

Specifies the offset of the control block identifier from the beginning of the control block. IDOFF= is required if you specify TYPE=.

### KEYOFF=

Specifies the offset from the beginning of the control block of the key field. KEYOFF= is required if you specify TYPE=.

### KEYLEN=

Specifies the length of the key field for the control block verification field that this macro builds. KEYLEN= is required if you specify TYPE=.

### MTTRFLD=

Specifies the offset of a field that contains a MTTR to verify.

**Note:** MTTRFLD= and MQTRFLD= are mutually exclusive.

### MQTRFLD=

Specifies the offset of a field that contains a MQTR to verify.

**Note:** MQTRFLD= and MTTRFLD= are mutually exclusive.

### ROUTINE=

Specifies the name of a control block-specific routine that JES2 calls for additional verification during an invocation of the \$VERIFY service.

The routine specified must be in the same CSECT/RSECT as the table being defined using \$VERTAB.

### TABLE=

Specifies the beginning (HASP) or end (END) of the verification table.

#### Note:

1. The table **must** start with a \$VERTAB TABLE=HASP.
2. The verification table is prefixed by a control block pool header.
3. TABLE= and TYPE= are mutually exclusive keywords. A warning message will be issued if the two are specified together, and the mapping DSECT will not be generated.
4. The table **must** end with \$VERTAB TABLE=END.

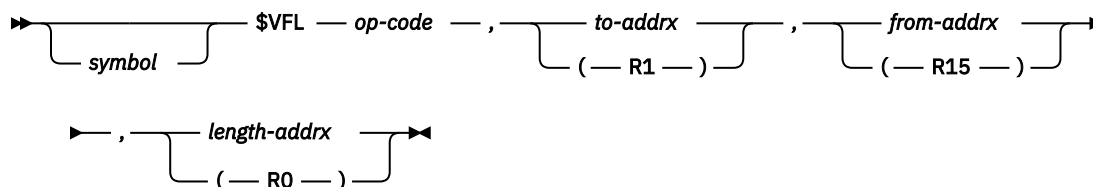
## Environment

- JES2 main task, user task, and HASPFSSM address space.
- \$WAIT is not applicable – this macro generates a DSECT or a static table entry; it does not generate executable code.

## \$VFL – Variable field length instruction operation

Use \$VFL to provide certain storage-to-storage operations where the field lengths exceed 256 bytes or where no assembler instructions exist.

## Format description



**op-code**

Specifies the storage-to-storage operation as one of the following:

**NC**

## And operation

XC

### Exclusive or operation

OC

Or operation

## MVC

Nondestructive overlapping move operation (see note 3)

**to-addrx**

Specifies the address of the first field (see note 1).

## from-addrx

Specifies the address of the second field (note 2).

**length-addrx**

Specifies the total number of bytes in the field (see note 1).

**Note:**

1. If the length operand is written as an address, the register contains the address of a fullword which contains the address of the field (which contains the field length).  
  
If the length operand is written using register notation, it represents the address of the field that contains the field length. If register notation is used, the address (or field length) must be loaded into the designated register before the execution of the macro instruction.
2. Condition codes from the execution of this macro are not usable.
3. When MVC is specified, a shift character long operation is performed. The number of bytes specified by length is moved from the *from* address to the *to* address. The origin and destination fields may overlap in any desired manner; the character string is moved intact without propagating the non-overlapping portion of the fields. \$VFL MVC is intended to be used in exceptional situations. For performance reasons, it should not be used where MVCL or an executed MVC would suffice.

## Environment

- Main task and subtask.

- User address space.
- \$WAIT cannot occur.

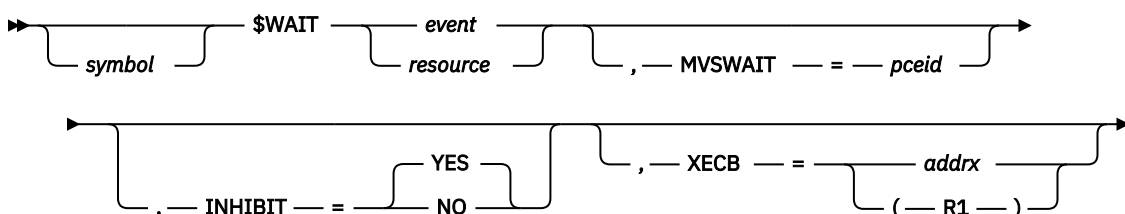
## \$WAIT – Wait for a JES2 event

Use \$WAIT to place the associated processor in a JES2 wait state and specify the event or resource for which the processor is waiting.

Optionally, use \$WAIT to specify an extended ECB structure (XECB) which may be posted by OS/VS service or some other task. If the XECB has already been posted, \$WAIT returns immediately to the processor; otherwise, \$WAIT initializes the extended ECB and places the processor in a JES2 wait state.

Callers in AR ASC mode are supported. Access registers are restored only if the caller is in AR ASC mode. The ASC mode is always restored upon return from \$WAIT.

### Format description



#### event | resource

Specifies the JES2 event or resource for which the processor is to wait as one of the following:

##### Event:

##### HOLD

Waiting for a \$S operator command

##### IO

Waiting for the completion of an input/output operation

##### OPER

Waiting to be reactivated

##### POST

Waiting for some resource or any \$POST

##### WORK

Waiting for more work

##### Resource:

##### ABIT

Waiting for the next dispatcher cycle.

##### ALICE

PCEs waiting for incomplete warm start.

##### ALOC

Waiting for allocation.

##### ARMS

Automatic restart manager support processor.

##### BERTL

Waiting for a BERT lock to free.

##### BERTW

Waiting for a free BERT.

##### BREG

PCEs waiting for WLM registration requests.

## **\$WAIT**

### **BUF**

Waiting for a JES2 buffer.

### **CCAN**

Cancel JOB/TSU/STC in conversion.

### **CKPT**

Waiting for the completion of a JES2 checkpoint.

### **CKPTL**

Looking for CKPT READ.

### **CKPTP**

Waiting for a checkpoint post.

### **CKPTW**

Waiting for checkpoint work.

### **CMB**

Waiting for a console message buffer.

### **CNVT**

Waiting for a converter.

### **DAWN**

Post PCEs waiting for work notifications.

### **DILBERT**

PCEs waiting for \$DILBERT requests.

### **EDSQ**

Email Delivery Services queue has changed status.

### **EOM**

Waiting for End Of Memory events.

### **FSS**

Waiting for completion of FSS-level processing.

### **GENL**

Provides a method of communication from one processor control element (PCE) to another. It does not provide serialization between the PCEs. You must ensure that the condition of the waiting PCE is satisfied before it is posted. Frequent use of the GENL resource name has a severe impact on your installation's performance.

### **HOMOG**

Within your MAS, either the member running the highest or lowest level of JES2 has changed. \$HCT fields, \$HIGHVER and \$LOWVER reflect these changes, respectively.

### **HOPE**

Waiting for an output processor.

### **IRCLEAN**

Internal Reader Cleanup needed.

### **IMAGE**

Waiting for a UCS or FCB image load completion.

### **JCMD**

Waiting for a job queue element (JQE) that needs to be canceled (\$C) or restarted (\$E).

### **JOB**

Waiting for a job.

### **JOE**

Waiting for a job output element (JOE) to be freed.

### **JOEI**

JOE index services.

### **JOT**

Waiting for job output table service.

**JQRB**

Process JQRB queue.

**LOCK**

Waiting for a lock.

**MAIN**

Waiting for storage.

**MFMT**

A spool volume has been mini-formatted. That is, JES2 has completed writing the 8-byte signature records for all track groups. Mini-formatting does not affect existing information on the spool.

**MLLM**

Line manager resource \$POSTs.

**NEWS**

PCE waiting for a JNEW update (part of JESNEWS process).

**NRM**

PCE checking the availability of NJE devices and initiating their starts appropriately.

**PCETM**

Waiting for resource manager to detach PCE.

**PSO**

PSO processor waiting for work.

**PURGE**

Purge processor is waiting for work.

**PURGS**

Waiting for purge resources from \$PURGER.

**RMWT**

Waiting for resource manager to finish processing.

**RSV**

A JES2 RESERVE has been satisfied.

**SMF**

Waiting for an SMF buffer.

**SPI**

PCEs waiting for SYSOUT API requests.

**SPIN**

A spin data set has been created.

**STAC**

STATUS/CANCEL resource type.

**TRACK**

Waiting for a direct-access track address.

**TRACP**

Waiting for a spool track group for a privileged job.

**UNIT**

Waiting for a device control table.

**WARM**

Warm processor is waiting for work.

**XMITJOB**

Waiting for a job to be transmitted to another node.

**INHIBIT=**

Specifies whether the processor issuing this macro instruction is to be dispatched if specifically posted (\$POST).

**YES**

All posts (\$POST) specifying this processor are ignored, except for the one indicating completion of the event specified in this macro instruction, or the one indicating the optional XECB has been POSTed.

**NO**

The processor issuing this macro instruction is to be dispatched if specifically posted (\$POST) for any event.

**MVSWAIT=**

Specifies whether the current PCE is to be tested against the PCEID specified here. If the PCE IDs do match, the current PCE and JES2 are placed in an MVS WAIT rather than a JES2 dispatcher \$WAIT. If MVSWAIT= is specified and the current PCE does not match the *pceid* that is specified here, then a JES2 \$WAIT is executed. A JES2 \$WAIT is generated for the PCE if MVSWAIT= is not specified. If you specify MVSWAIT=, you must also specify XECB=.

**XECB=**

Specifies the address of an XECB. The processor issuing this macro instruction is dispatched when the XECB is posted, or immediately resumes control if the XECB has already been posted. If register notation is used, the designated register must be loaded with the address of the XECB before executing this macro.

**Note:**

1. The execution of this macro instruction requires register 15; also, register 1 is required if the XECB option is used.
2. The JES2 processor is dispatched if either the JES2 event/resource is posted or the ECB in the XECB control block is posted.

**Attention:**

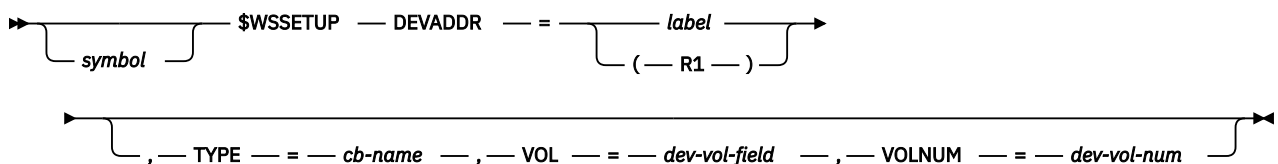
- If the XECB option is used, the processor is dispatched by either the JES2 event occurrence or the POST of the XECB.
- You may clear the first word of the XECB; clearing the entire XECB causes problems in other JES2 chains that might be in use.

**Environment**

- Main task and JES2 initialization.
- \$WAIT is not applicable.
- MVS WAIT can occur.
- Callers in AR ASC mode are supported.

**\$WSSETUP – Set values required for work selection**

Use the \$WSSETUP macro instruction to set those values that are required to support work selection.

**Format description****DEVADDR=**

Specifies the address of a device control table (DCT), a partitioned information table (PIT), or a work selection parameter (WSP). Specify this address either by a label or a register; the address is loaded in register 1.



**TYPE=**

Specifies the device control block name used to calculate the offset for the fields specified by the VOL= and VOLNUM= keywords. If this keyword is specified, both VOL= and VOLNUM= must also be specified.

**VOL=**

Specifies the device's volume field. The offset for this field is calculated using the name specified by the TYPE= keyword. If this keyword is specified, both TYPE= and VOLNUM= must also be specified.

**VOLNUM=**

Specifies the volume number field. The offset for this field is calculated using the name specified by the TYPE= keyword. If this keyword is specified, both TYPE= and VOL= must also be specified.

## Environment

- JES2 main task.
- \$WAIT cannot occur.

## \$WSTAB – Map and generate the work selection table entries

Use the \$WSTAB macro instruction to map and generate the work selection table entries.

Use \$WSTAB to map and generate WS table entries.

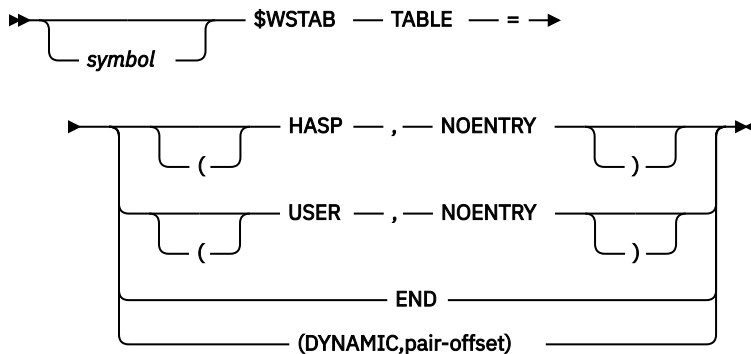
\$WSTAB entries are used to define the start of a user table (\$WSTAB TABLE=USER...) or a JES2 table (\$WSTAB TABLE=HASP...), the end of a table (\$WSTAB TABLE=END) or an entry in a table (\$WSTAB NAME=MARK...).

**Note:** The format description that follows breaks the macro into a **boundary form** (the form that starts or ends a table) and an **entry form** (the form that defines each table entry).

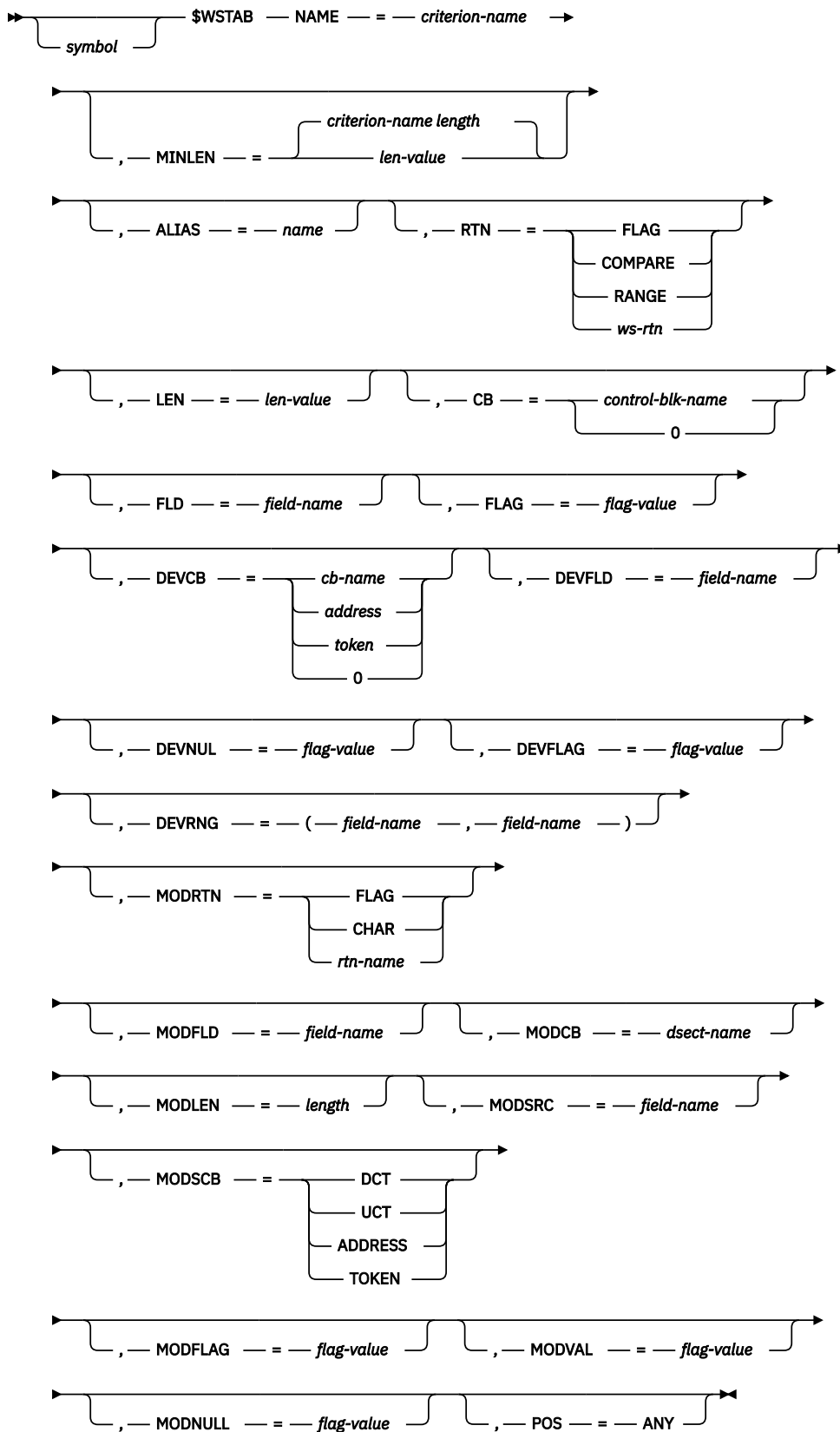
## Format description

The following format descriptions apply:

### Boundary form



## Entry form



### NAME=

Specifies a 1-8 character name for an individual work selection criterion.

**MINLEN=**

Specifies the minimum length that is acceptable for the criterion name specified on the NAME= keyword. This keyword does not also support the ALIAS= keyword. If this keyword is not specified, the length defaults to the length of the ws-name specified by the NAME= keyword.

**ALIAS=**

Specifies an alternate (alias) 1- to 4-byte character name for the work selection criterion. MINLEN= does not support this keyword.

**RTN=**

Specifies the name of the routine used to check the comparison field against the device field. This keyword is required.

**FLAG**

Indicates to call the general flag routine for this criterion during work selection.

**COMPARE**

Indicates to call the general compare routine for this criterion during work selection.

**RANGE**

Indicates to call the general range routine for this criterion during work selection.

**ws-name**

Specifies any other valid work selection routine that is to be called.

The following are the register values on entry to the specified routine:

**R0:**

N/A

**R1:**

N/A

**R2:**

N/A

**R3:**

WSA address

**R4:**

Pointer to current mask byte.

**R5:**

WSTAB address.

**R6:**

N/A

**R7:**

Comparison length.

**R8:**

DEVFLD= field address (DEVCB address if DEVFLD= is not specified).

**R9:**

N/A

**R10:**

FLD= field address (CB address if FLD= is not specified)

**R11:**

HCT address..

**R12:**

N/A

**R13:**

PCE address.

**R14:**

Return address

**R15:**

Routine address.

The following are the register values and conditions on exit from the specified routine:

**R0:**

May be destroyed.

**R1:**

May be destroyed.

**R2:**

**Must be preserved.**

**R3:**

**Must be preserved**

**R4:**

**Must be preserved**

**R5:**

May be destroyed.

**R6:**

May be destroyed.

**R7:**

May be destroyed.

**R8:**

May be destroyed.

**R9:**

**Must be preserved**

**R10:**

May be destroyed.

**R11:**

**Must be preserved**

**R12:**

**Must be preserved**

**R13:**

**Must be preserved**

**R14:**

May be destroyed.

**R15:**

Return Code.

The following return codes (in decimal) are returned in register 15.

**Return Code**

**Meaning**

**0**

Reject work.

**4**

Criterion matched, continue checking additional criteria.

**8**

Accept work (no further checks).

**12**

Reject work if prior to "/", Continue checking if after "/".

**Note:**

1. \$SAVE/\$RETURN may be used to preserve registers but may affect performance of devices specifying this criterion.
2. \$WAIT must not be done by this routine.

**CB=**

Specifies the control block name required to resolve the field specified by the FLD= keyword. This keyword is required. The valid control block names follow.

**Control Block****Meaning****JQE**

JQE required

**WJOE**

Work JOE required

**CJOE**

Character JOE required

**JOA**

Artificial JOE required

**HCT**

HCT required

**NJHO**

Spool offload header section

**NJHOX**

Affinity section of job header required

**NJHG**

General section of job header required

**NJH2**

JES2 section of job header required

**NJHT**

Security section of job header required

**NJHU**

User section of job header required

**NDHG**

General section of data set header required

**NDHA**

3800 Printer section of data set header required

**NDHS**

Data stream section of data set header required

**NDHT**

Security section of data set header required

**NDHU**

User section of data set header required

**WSA**

Work selection area required

**ZERO**

No control block required for specified criterion. If CB=ZERO is specified, both FLD= and FLAG= keywords (if also specified) are ignored.

**(TOKEN,name)**

- (TOKEN,name,SYSTEM)
- (TOKEN,name,SUBSYS)
- (TOKEN,name,HOME)

- (TOKEN,name,PRIMARY)
- (TOKEN,name,TASK)

Specifies the NAME associated with a name/token pair (created using the \$TOKENSR service). The name/token pair contains the address of the control block; the control block contains the DEVFLD or DEVRNG fields. NAMES can be up to 16 bytes long, and must match the name specified on a \$TOKENSR call. The first 4 bytes of the token are assumed to be the control block address. The remaining 12 bytes of the token are not used.

**FLD=**

Specifies the name of the work selection comparison field against which the device field is compared. If the FLAG= keyword is specified, this keyword is used as a flag byte to be compared against the flag byte setting specified by FLAG=.

**FLAG=**

Specifies the label of the field in the FLD= byte to be tested under mask for work selection. If FLAG= is specified, FLD= must also be specified.

**LEN=**

Specifies the length of a character comparison between a control block field and a device field. This keyword defaults to the length of the comparison field specified by the FLD= keyword.

**TABLE=**

Specifies the start or end of a work selection table.

Specify TABLE=HASP or TABLE=USER to start the corresponding table, and optionally a second parameter of NOENTRY (e.g. TABLE=(USER,NOENTRY)) to indicate no ENTRY statement need be generated for the label of the scan table.

DYNAMIC specifies that this is a DYNAMIC table. The second and subsequent positionals, *pair-offset*, specify the offset of the \$PAIR in either the MCT or UCT control block with which this table is to be associated. If the table pair in the MCT or UCT is not defined through the \$PAIR macro, an assembler MNOTE will be issued.

Specify TABLE=END to terminate a scan table.

Other operands are ignored if TABLE is specified, and a label is required on the \$WSTAB macro if a table is being started. If TABLE= and NAME= are both not specified, only the mapping of an \$WSTAB entry is generated by the macro.

**DEVCB=**

Specifies the control block required to resolve the field specified by the DEVFLD operand. The values that can be specified are as follows.

**Control Block  
Meaning**

**DCT**  
DCT required

**PIT**  
PIT required

**HCT**  
HCT required

**UCT**  
UCT required

**(ADDR,value)**

Value is a label in the current module which is control block address. The control block address (from the token) is combined with the field and dsect specified on DEVFLD= or DEVRNG= key word to determine the field address.

**(TOKEN,name)**

name specifies the TOKEN name, and level is one of the values accepted by the \$TOKENSR macro. TASK level is the default.

- (TOKEN,name,SYSTEM)
- (TOKEN,name,SUBSYS)
- (TOKEN,name,HOME)
- (TOKEN,name,PRIMARY)
- (TOKEN,name,TASK)

Specifies the NAME associated with a name/token pair (created using the IEANTCR callable service) which contains the address of the control block that contains the DEVFLD or DEVRNG fields. NAMES can be up to 16 bytes long (and must match the name specified on a IEANTCR call). The first 4 bytes of the token are assumed to be the control block address. The remaining 12 bytes of the token are not used.

The control block address (from the token) is combined with the field and dsect specified on DEVFLD= or DEVRNG= keyword to determine the field address. Token name on second operand of DEVCB= is required and only allowed, if TOKEN is specified on first operand of DEVCB= keyword.

NAME/TOKEN pairs can be created at various levels. JES2 supports tokens at the SYSTEM or TASK level. The third operand on DEVCB= specifies the level passed on the IEANTRT service call.

### **ZERO**

No control block is needed for this criterion.

If 0 is specified, DEVFLD and DEVFLAG will be ignored. If a general purpose RTN= is specified (CHAR, RANGE, or FLAG), the effect of specifying 0 is as follows:

#### **CHAR**

Always matches.

#### **RANGE**

Always matches.

#### **FLAG**

Matches if FLAG bits in CB are off (as if DEVFLG was also 0). FLAG is a required operand.

### **DEVFLD=**

Specifies the name of the device field against which work selection comparison field (FLD=) is compared. If the DEVFLAG= keyword is specified, this keyword is used as a flag byte to be compared against the flag byte setting specified by DEVFLD=. Also, if DEVFLD= is specified, DEVRNG= cannot also be specified.

### **DEVFLAG=**

Specifies the label in the DEVFLD= byte to be tested under mask for work selection. If DEVFLAG= is specified, DEVRNG= cannot also be specified.

### **DEVRNG=**

Specifies the names of the upper and lower device fields against which the control block field (as specified by the CB= and FLD= keywords) is compared. If either DEVFLAG= or DEVFLD= are specified, this keyword cannot also be specified.

### **DEVNUL=**

Specifies the value for the device flag byte that is used to determine if a null value was specified for this criterion.

### **MODRTN=**

Specifies the name of the routine used to modify the criterion following selection by a offload receiver. This keyword is required if the criterion is to be modified.

#### **FLAG**

Indicates that the general flag routine is to be called

#### **CHAR**

Indicates that the general character routine is to be called

#### **rtm-name**

Indicates a valid modify routine that is to be called

**MODFLD=**

Specifies the name of the field that is to be modified when the job or SYSOUT is reloaded.

**MODCB=**

Specifies the DSECT name required to resolve the field specified by the MODFLD= keyword. Valid field names are as follows:

**Field Name****Meaning****NJHG**

General section of the job header

**NJHO**

Spool offload header section

**NJH2**

JES2 section of the job header

**NJHU**

User section of the job header

**NDHG**

General section of the data set header

**NJHA**

3800 printer section of the data set header

**NDHS**

Data stream section of the data set header

**NDHU**

User section of the data set header

**MODLEN=**

Specifies the length of the field that is to be modified. If this length is not specified, it defaults to the length of the name specified by the MODFLD= keyword.

**MODSRC=**

Specifies the name of the field in the control block indicated by the MODSCB= keyword that contains the value that replaces the current value in the field specified by the MODFLD= keyword. If MODRTN= is set to either FLAG or CHAR this keyword is required.

**MODSCB=**

Specifies the control block containing the MODSRC field. Valid values for this parameter are the following:

**DCT**

Get value from DCT field.

**UCT**

Get value from UCT field.

**(TOKEN,name)**

name specifies the TOKEN name, and level is one of the values accepted by the \$TOKENSR macro. TASK level is the default.

- (TOKEN,name,SYSTEM)
- (TOKEN,name,SUBSYS)
- (TOKEN,name,HOME)
- (TOKEN,name,PRIMARY)
- (TOKEN,name,TASK)

Specifies the NAME associated with a name/token pair (created using the IEANTCR callable service) which contains the address of the control block that contains the MODSRC field. NAMES can be up to 16 bytes long and must match the name specified on a IEANTCR call. The first 4 bytes of the token are assumed to be the control block address. The remaining 12 bytes of the token are not used.



The control block address from the token is combined with the field and dsect specified on MODSRC= keyword to determine the field address. Token name on second operand of MODSCB= is required and only allowed, if TOKEN is specified on first operand of MODSCB= keyword.

NAME/TOKEN pairs can be created at various levels. JES2 supports tokens at the SYSTEM or TASK level. The third operand on MODSCB= specifies the level passed on the IEANTRT service call.

**MODFLAG=**

Specifies the flag value to be set in MODFLD if MODRTN=FLAG is specified. This keyword is required if MODRTN=FLAG is specified.

**MODVAL=**

If MODRTN=FLAG is specified, this keyword specifies a mask that is compared against the byte specified by the MODSRC= keyword. If the flag is set, then the flag specified by the MODFLAG= keyword is turned off in the byte specified by the MODFLD= keyword. If the flag is not set, then the flag specified by the MODFLAG= keyword is turned off in the byte specified by MODFLD=. This keyword is required if MODRTN= is specified.

**Note:** MODVAL= and MODNULL= must both map to the same MODFLD= byte.

**MODNULL=**

If MODRTN=FLAG is specified, this keyword specifies a mask that is compared against the value specified by the MODSRC= keyword. If the null flag is set, then the device characteristic in the MOD= list has previously been set to NULL and is not modified.

**Note:** MODNULL= and MODFLD= must both map to the same MODFLD= byte.

**POS=**

If POS=ANY is specified, this entry describes a work selection criterion which has the same significance, whether it was added before or after the '/' character.

## Environment

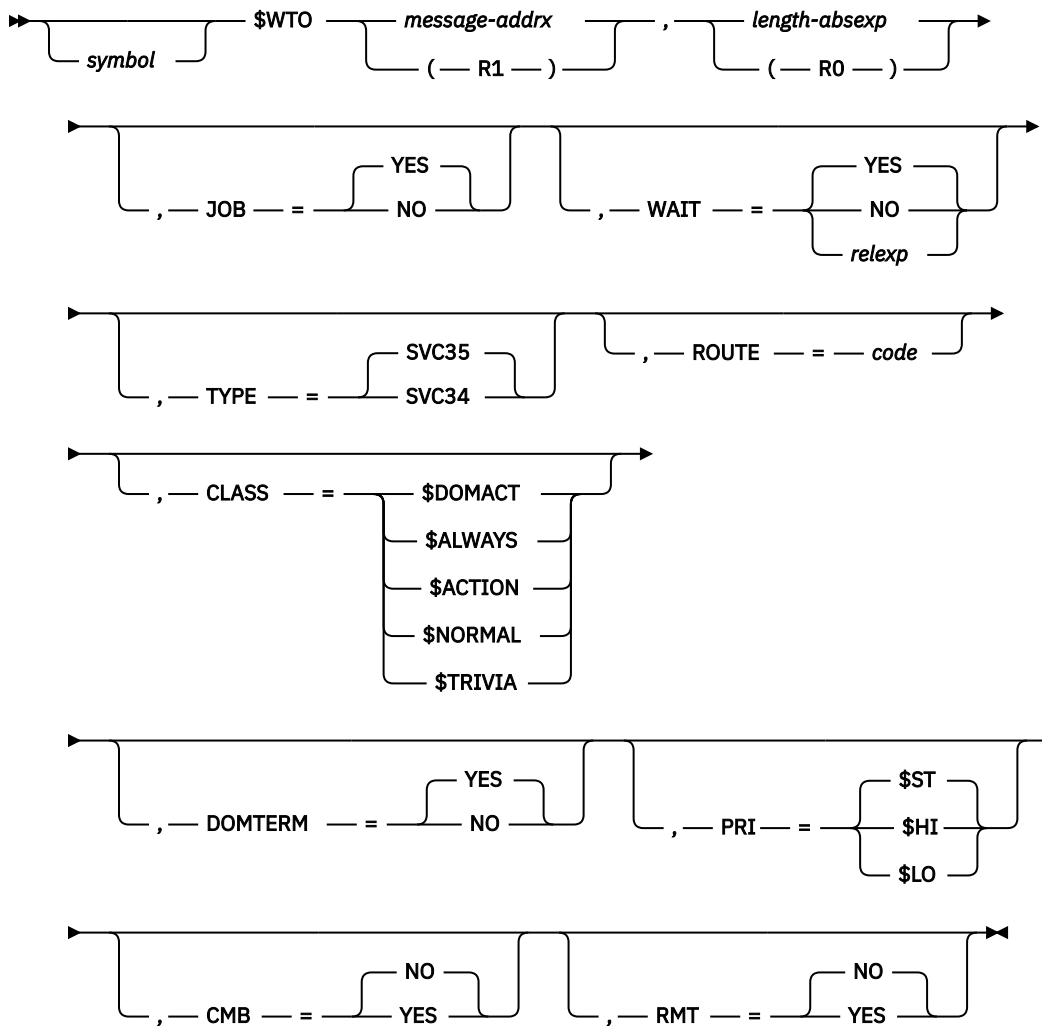
- JES2 main task.
- \$WAIT cannot occur in a routine specified by either the RTN= or MODRTN= keywords.

## \$WTO – JES2 Write to operator – JES2 environment

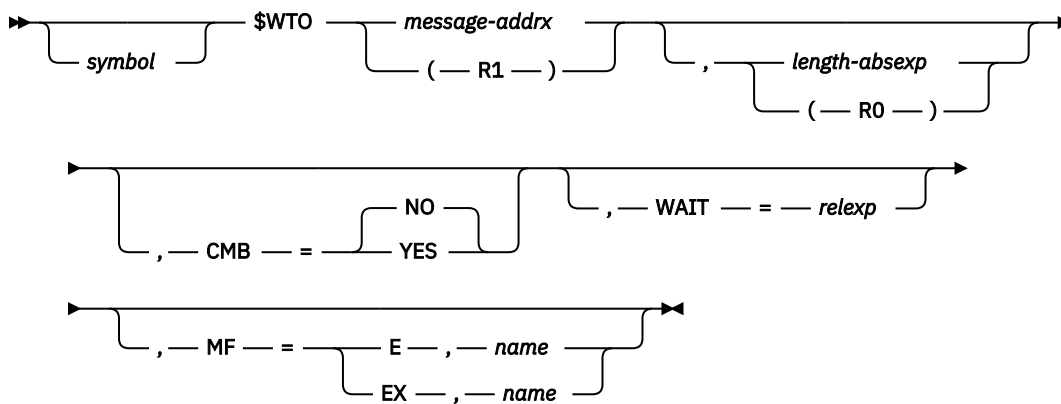
---

Use \$WTO to initiate the display of a message intended for the operator either on one or more operating system consoles or a JES2 remote work station console or printer device.

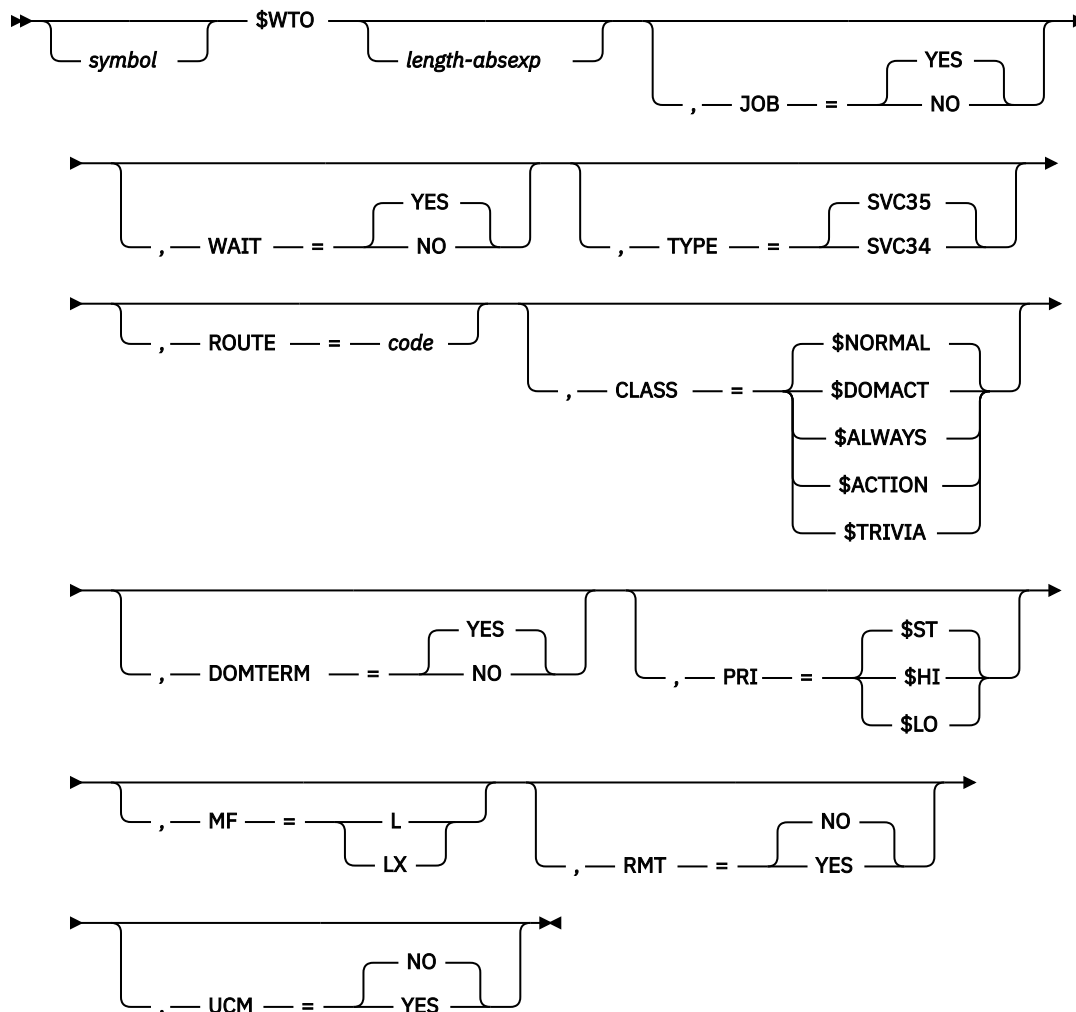
## Format description – Standard form



## Format description – Execution form



## Format description – List form



### message

Specifies the address of a message which is to be displayed on the designated consoles or the address of a console message buffer (CMB) where the message resides if CMB=YES is specified. If register notation is used, the address must be loaded into the designated register before the execution of this macro instruction.

If you specify TYPE=SVC35, or allow TYPE= to default, the message must have the same format as a message generated by the \$MSG macro.

### length

Specifies the length of the above message. If register notation is used, the value must be loaded into the rightmost byte of the register (R0) before the execution of the macro instruction. The rest of the register must be 0 unless the message is being sent to a remote terminal (see RMT operand). The maximum message length is 125 bytes if JOB=NO is specified and 116 bytes if JOB=YES is specified.

**Note:** When using the execute and list forms of the macro instruction, the length must be specified in one of the macro instructions but not both.

### CLASS=

Specifies the class of the message as one of the following:

#### \$DOMACT

The message requires immediate action and is always written. (MCS descriptor code 2)

## **\$WTO**

The \$DOMACT specification is reserved for \$WTOs issued to logical consoles. On return from \$WTO processing, register 1 contains the address of the console message buffer (CMB) containing the message. The CMB is retained in the system until a corresponding \$DOM is executed using the returned pointer.

### **\$ALWAYS**

The message is essential and is always written.

**Note:** Issue \$DOM to empty a full console message buffer (CMB).

### **\$ACTION**

The message requires eventual operator action.

### **\$NORMAL**

The message is considered important to normal computer operations. (MCS descriptor code 4)

### **\$TRIVIA**

The message is considered unimportant to normal computer operations.

If you omit this operand, \$NORMAL is assumed.

### **CMB=**

Specifies whether the user of the \$WTO macro instruction has provided a console message buffer for use by console services as follows:

#### **NO**

A console message buffer has not been provided, and the message operand refers to message text.

#### **YES**

A console message buffer has been provided, and the message operand refers to the console message buffer containing the user's message.

If this operand is omitted, CMB=NO is assumed.

#### **Note:**

1. If CMB=YES is specified, the message must appear in the appropriate locations within the console message buffer as follows:

##### **JOB=YES**

Message starts in CMBTEXT field

##### **JOB=NO**

Message starts in CMBJOB field

2. If CMB=YES is specified, the CMB must have been obtained with the \$GETCMB macro instruction.
3. Use the \$MSG macro instruction to generate the message text or, if dynamically generated, at least the message identification section.

### **DOMTERM=**

Enables you to specify whether an outstanding action message is to be deleted when JES2 terminates.

#### **YES**

To delete this outstanding action message when JES2 terminates, code DOMTERM=YES or omit DOMTERM=. This applies to messages issued with CLASS=\$DOMACT.

When issuing a message for which DOMTERM=YES has been specified, JES2 uses descriptor code 7 in addition to currently used descriptor codes.

YES is the default.

#### **NO**

If you want this outstanding action message to still be displayed after JES2 terminates, code DOMTERM=NO.

**JOB=**

Specifies whether the job identification and job name from the job control table (JCT) or the job queue element (JQE) are appended to the start of the messages as follows:

**YES**

The job information is inserted in the message and passed to WTO.

**NO**

The job information is not inserted in the message.

If this operand is omitted, JOB=YES is assumed.

**Note:** If JOB=YES is specified, register 10 must be loaded with the address of either the job control table entry, the job queue element, or zero (0) before the execution of this macro instruction, or the job information printed is unpredictable. When R10 is zero (0), the JQE in PCEJQE will be used to obtain job information.

**PRI=**

Specifies the priority of the message as one of the following:

**\$HI**

High priority

**\$ST**

Standard priority

**\$LO**

Low priority

If this operand is omitted, \$ST priority is assumed.

**RMT=**

Specifies whether the display location is to be a JES2 remote work station.

**NO**

The destination is not a remote work station and therefore must be logical routing or UCM.

**YES**

The destination is a work station, and the following action is required before executing execution forms of the macro instruction:

- For standard or MF=E forms, the remote number is set in register 0 (R0); bits 32-39 must be 0, the remote number must be set into bits 40-55, and the length of the message must be set into bits 56-63. You must specify R0 in the executing macro instruction's length operand.
- For MF=EX form, you must place the remote number in the MF=LX halfword field corresponding to CMBRMT field of the CMB.

**ROUTE=**

When using standard and MF=L forms, specifies the JES2 logical routings that are converted to operating system equivalent routings as follows:

**Designation****Console Specified****\$LOG**

Hard-copy console (MCS route code 0)

**\$ERR**

Error consoles (MCS route code 10)

**\$UR**

Unit-record operations area (MCS route code 7)

**\$TP**

Teleprocessing operations area (MCS route code 8)

**\$TAPE**

Tape operations area (MCS route codes 3,4,5 and 6)

**\$MAIN**

Chief operator's area (MCS route codes 1 and 2)

**\$ALL**

All of the above consoles

**0**

JES2 automatically determines route codes (if needed) based on the message type.

**Note:** You can specify multiple values, for example, ROUTE=\$LOG+\$UR.

When using the MF=LX format, unless UCM=YES or RMT=YES is specified, the operating system console routings must be specified directly using the X'xxxx' or B'xxxxxxxxxxxxxxxx' form.

If this operand is omitted, the \$LOG console is assumed unless UCM=YES or RMT=YES is specified.

**Note:** The name \$ALL should not be used with any ROUTE= values. It should only be specified alone; otherwise, results are unpredictable.

**TYPE=**

Specifies the logical meaning of the ROUTE operand as follows:

**SVC35**

The message is to be displayed on the dynamically designated remote work station console or the operating system consoles that have been set to receive the message category.

**Note:**

1. If this option is selected, use only one of the following specifications: ROUTE=, UCM=, or RMT=.
2. If you specify TYPE=SVC35, or allow TYPE= to default, the message must have the same format as a message generated by the \$MSG macro.

**SVC34**

The message is to be given to the operating system as a command. If this option is selected, ROUTE=, UCM=, and RMT= have no meaning.

**WAIT=**

Specifies the action to be taken if console message buffers are not available as follows:

**YES**

Return is not made until a console message buffer has become available and the message has been queued.

If you code WAIT=YES or omit WAIT, exit 10, when it receives control, will be told that it can take an action that will result in a \$WAIT.

**NO**

An immediate return is made with the condition code set as follows:

**CC=0**

No console message buffers are available. The message was not accepted and the macro instruction must be reissued.

**CC≠0**

The message was accepted.

**(NO,DEMAND)**

Return will not be made without queueing message. The console message buffer will GETMAINED if required.

**relexp**

A location to which control is returned if CMB=YES is not specified and no console message buffers are available.

If this operand is omitted, WAIT=YES is assumed.

**Note:**

1. The specification of WAIT=NO has no meaning when the console message buffer is provided by the user and CMB=YES is specified.
2. The specification of WAIT=relexp has no meaning if MF=L or LX.

**UCM=**

Specifies whether the display location is to be a specific operating system console identified by a console ID.

**NO**

The destination is not a specific console and therefore must be logical routing or RMT.

**YES**

The destination is a console ID set in CMBUCMID of the CMB DSECT.

**Note:** You must use the MF=LX form of this macro, and you must set the 4-byte console ID in the CMBUCMID field of the macro parameter list.

**Note:**

1. MF=L specifies the list form of the \$WTO macro instruction. MF=L or MF=E are forms of the \$WTO macro instruction that allow the predefinition of parameter lists similar to MVS supervisor services macros specified with the “execute” or “list” forms.
2. MF=LX specifies the extended list form of the \$WTO macro instruction. MF=(E,name) specifies the extended execute form of the \$WTO macro instruction using a remote control program parameter list. MF=(EX,name) specifies the execute form of the \$WTO macro instruction using a remote control program parameter list.
3. MF=LX or MF=EX are extended forms of MF=L or MF=E, respectively. You can use these extended forms to specify the \$WTO macro instruction without using \$GETCMB, formatting the CMB in detail, and using \$WTO CMB=YES.
4. The CMBTO field in the CMB can be filled out to route messages to another node in the JES2 network or to another member in the multi-access spool JES2 complex. Field CMBUCMID can be set to indicate the 4-byte console id. CMBUCMA should be set to indicate the console area. MLWTOs can be used with explicit route and descriptor codes. To do this, you must format the console area with the CMB mapping and the MF=EX form of the \$WTO macro instruction.
5. The MF=LX form of the \$WTO macro instruction only allows explicit route codes through ROUTE=B'xxxxxxxx...' or X'xxxx'. Descriptor codes cannot be specified using the MF=LX form of the \$WTO macro.
6. The CMB mapping starts with the CMBFLAG field and **not** the CMB DSECT field.
7. ROUTE=\$LOG with no other route codes instructs JES2 to route messages to hardcopy only.

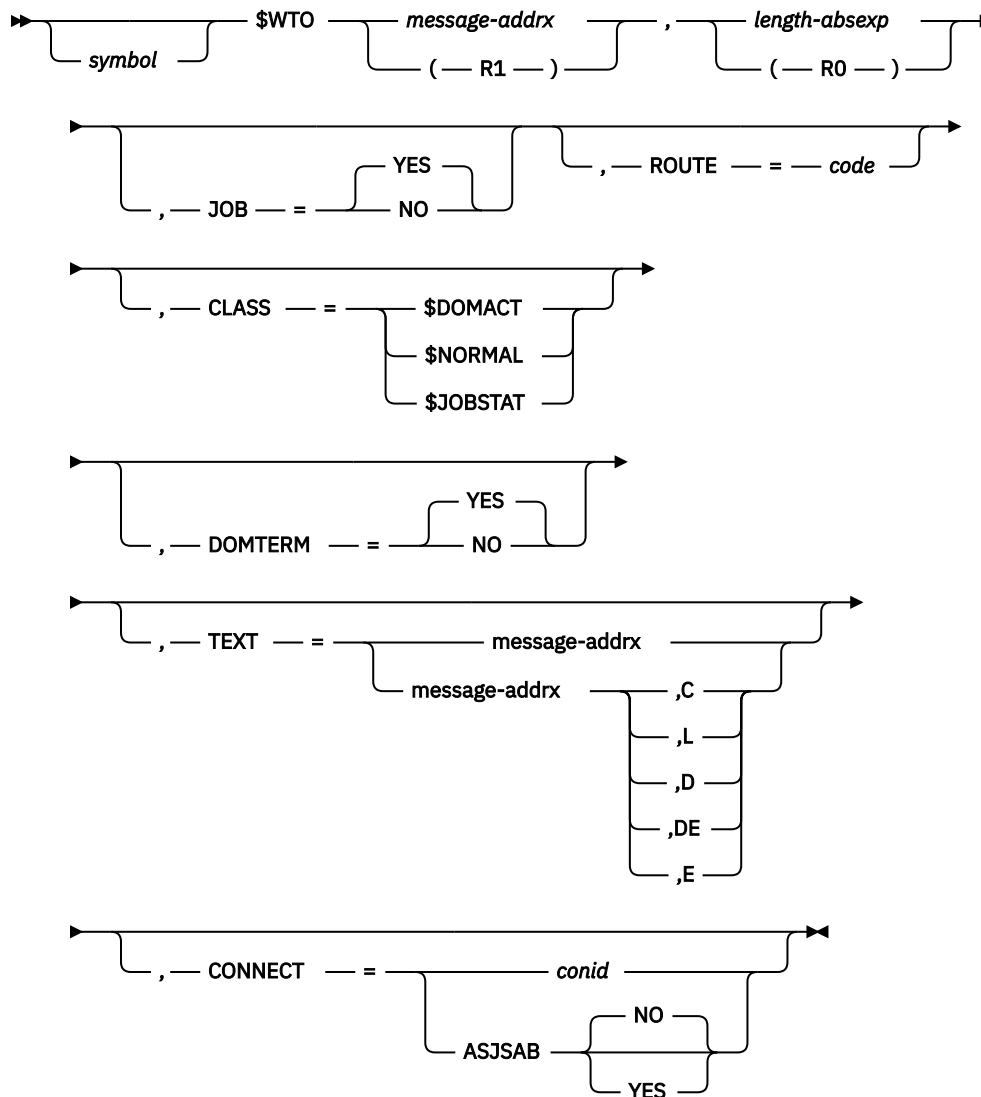
**Environment**

- Main task.
- \$WAIT can occur depending on how WAIT= is specified.

**\$WTO – JES2 Write to operator – User and subtask environment**

Use \$WTO to initiate the display of a message intended for the operator either on one or more operating system consoles or a JES2 remote work station console or printer device.

## Format description – Standard form



### message

Specifies the address of a message that is to be displayed on the designated consoles. If register notation is used, the address must be loaded into the designated register before the execution of this macro instruction. If the message begins with \$HASP, the \$ will be replaced by the value specified on the CONCHAR= parameter of the CONDEF initialization statement. Do not use \$MSG to generate messages issued with the \$WTO macro in the user or subtask environment.

### length

Specifies the length of the above message. If register notation is used, the value must be loaded into the rightmost byte of the register (R0) before the execution of the macro instruction. The rest of the register must be 0. The maximum message length is 125 bytes if JOB=NO and 116 bytes if JOB=YES.

### CLASS=

Specifies the class of the message as one of the following:

#### \$DOMACT

The message requires immediate action and is always written. (MCS descriptor code 2)

On return from \$WTO processing, register 1 contains the DOMID. The \$WTO caller is responsible for issuing the DOM macro to delete the message using the DOMID passed back.

#### \$NORMAL

The message is considered important to normal computer operations. (MCS descriptor code 4)



**\$JOBSTAT**

Descriptor code is set to MCSJOBST. (MCS descriptor code 6)

If you omit this operand, \$NORMAL is assumed.

**CONNECT=**

If the second or subsequent line of a multi-line WTO, specifies the MLWTO connect ID. Only valid if TEXT= was specified.

**ASJSAB=NO|YES**

Specify YES if the message is associated with the address space level unit of work. If NO, the default, the JSAB used will be the one pointed to by the STCB. If no JSAB is pointed to by the STCB, the ASSBJSAB will be used. If YES, then use ASSBJSAB.

**DOMTERM=**

Enables you to specify whether an outstanding action message is to be deleted after the issuing task terminates. DOMTERM= is meaning only if CLASS=\$DOMACT.

**YES**

Indicates that you want the outstanding action message deleted when the issuing task terminates. This applies to messages issued with CLASS=\$DOMACT. When issuing a message for which DOMTERM=YES has been specified, JES2 uses MCS descriptor code 7 in addition to currently used descriptor codes. YES is the default.

**NO**

Indicates that this outstanding action message should still be displayed after the issuing task terminates.

**JOB=**

Specifies whether the job identification and job name from the JSAB (or ASCB if no JSAB) are inserted in the message and passed to WTO.

**Note:** If you are in input services (internal reader or NETSRV) and want to use the job information for the submitted job, use \$BLDMSG with the JOB= specification instead of \$WTO.

**YES**

The job information is inserted in the message and passed to WTO. The jobname is inserted in the message only if it begins with "\$HASP" and then is inserted into position 10 of the message between the \$HASP $nnn$  and the message text.

**NO**

The job information is not inserted in the message.

If this operand is omitted, JOB=YES is assumed.

**ROUTE=**

Specifies the JES2 logical routings that are converted to operating system equivalent routings as follows:

**Designation****Console Specified****\$LOG**

Hard-copy console (MCS route code 0)

**\$ERR**

Error consoles (MCS route code 10)

**\$MCINFO**

Master console information (MCS route code 2)

**\$PGINFO**

Programmer information (MCS route code 11)

**0**

Hard copy only (MCS route 0)

**Note:** You can specify multiple values, for example, ROUTE=\$LOG+\$PGINFO.

If this operand is omitted it defaults to \$LOG.

**TEXT=**

Specifies the address of a half-word length followed by the text of the operator message. The address can be in a register (2-12) or be the name of a field. The length does not include the length of the half-word containing the length. The maximum message length is 125 bytes if JOB=NO and 116 bytes if JOB=YES.

If the message begins with "\$HASP" the \$ will be replaced by the value specified on the CONCHAR= parameter of the CONDEF initialization statement.

A second optional value is the line type for multi-line WTOs. Valid values are C, L, D, DE, and E. See the WTO macro for complete descriptions.

The connect id is returned in register 1. This is for specification in CONNECT= on subsequent lines of a multi-line WTO.

**Note:**

1. ROUTE=\$LOG with no other route codes instructs JES2 to route message to hardcopy only.
2. The positional message and length operands are mutually exclusive with TEXT=.

**Environment**

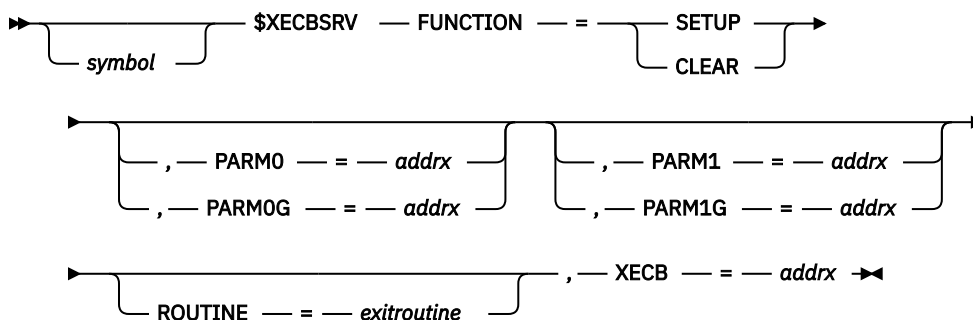
- User or subtask.

## **\$XECBSRV – Interface for extended event control block (XECB) services**

This macro provides an interface to JES2 services for manipulating extended ECB. Extended ECBs come in two forms. First form includes extended ECBs (XECB) used by the main task, and allows a PCE to be processed by \$POST when the XECB is posted. The second form allows a service routine to be run when an ECB is posted. Either form can be processed by this macro.

For FUNCTION=SETUP, when no routine is specified, the ECB is set to \$POST the calling PCE when the ECB is MVS posted. This form is only valid in the JES2 main task.

For FUNCTION=SETUP, if ROUTINE= is specified, that routine will get control with the parms specified.

**Format description****FUNCTION=**

Specifies the function requested.

**SETUP**

When no ROUTINE= is specified, set up an ECB so that a POST of the ECB will result in the \$POSTing of the current PCE. Upon return, the caller needs to check to determine if the ECB is posted while being setup.

When ROUTINE= is specified, set up the ECB so that the routine is called when the ECB is posted. On entry to the routine, the registers are as follows:

**R0**  
PARM0/PARM0G value

**R1**  
PARM1/PARM1G value

**R2**  
XECB address

**R3 - R10**  
N/A

**R11**  
HCCT address

**R12**  
N/A

**R13**  
N/A (NOT A SAVE AREA)

**R14**  
Return address

**R15**  
Entry address

#### **CLEAR**

Removes an XECB you previously setup or for which you issued a \$WAIT from all JES2 work chains. Use this function if your routine no longer requires the XECB.

#### **PARM0**

32 bit data area to pass to exit routine in register 0.

#### **PARM1**

32 bit data area to pass to exit routine in register 1.

#### **PARM0G**

64 bit data area to pass to exit routine in register 0.

#### **PARM1G**

64 bit data area to pass to exit routine in register 1.

**Note:** (PARM0= and PARM0G=) and (PARM1= and PARM1G=) are mutually exclusive.

#### **ROUTINE=**

Routine to call when ECB is posted.

#### **XECB=**

The address of the XECB to setup or clear.

This is a required parameter.

## **Environment**

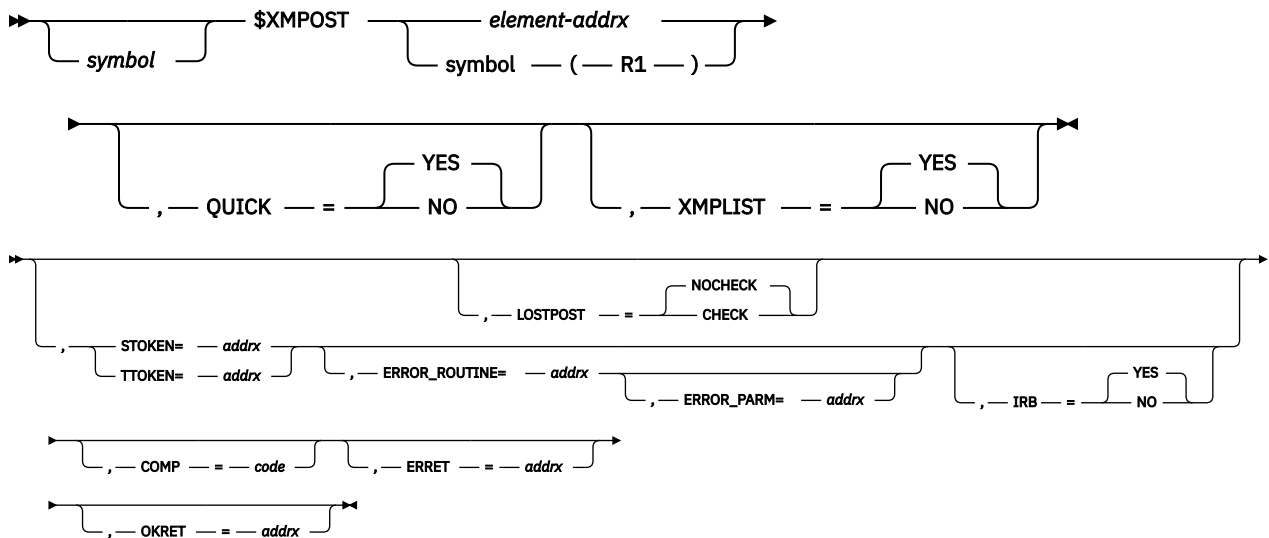
- JES2 main task if ROUTINE= is not specified.
- Any environment if ROUTINE= is specified.
- \$WAIT can occur.

## **\$XMPOST – POST task in another address space**

\$XMPOST provides the linkage to the cross-memory post service routine. There are 2 ways to invoke this macro. The traditional method uses a cross memory post parameter list (with an ECB address, ASCB address, and error routine address) and the newer method passing an STOKEN or a TTOKEN of the space

that you want to post. The newer method is preferred and verifies that the space and optionally task still exists before issuing the post.

## Format description



### Operands when STOKEN= and TTOKEN are not specified:

#### element

##### If XMPLIST=NO

Specifies a register that contains the address of an ECB that is to be cross-memory posted, or specifies the actual ECB itself.

##### If XMPLIST=YES

Specifies the address of a 3-word POST element formatted as follows:

##### +0

Address of error return.

##### +4

Address of ECB to be POSTed.

##### +8

Address of related ASCB.

##### If LOSTPOST=CHECK:

##### +C

ECB to be posted

#### QUICK=YES|NO

Specifies whether (YES) or not (NO) JES2 should attempt a “quick post” of the event control block (ECB).

**Note:** A quick post of an ECB causes JES2 to attempt to set the POST flag in the ECB before JES2 does a \$WAIT for the ECB. If you require a quick post, be certain that the ECB resides in addressable storage (that is, the ECB must be in either common storage or private storage of the same address space).

#### XMPLIST=YES|NO

Specifies whether the value that is specified for the first positional parameter is a cross memory parameter list (YES) or an ECB (NO). If XMPLIST=NO is coded, the asid to be posted is assumed to be that of JES2. The default for this parameter is YES.

**LOSTPOST=CHECK |NOCHECK**

Specifies whether (CHECK) or not (NOCHECK) JES2 should attempt to detect if a cross-memory post was lost.

**LOSTPOST=CHECK |NOCHECK**

Can only be specified if 'QUICK=YES' is specified (or defaulted) and 'XMPLIST=YES' is specified (or defaulted).

**Operands when STOKEN= or TTOKEN are specified:****element**

Specifies the ECB that is to be cross-memory posted. This cannot be passed in register R0 or R1.

**STOKEN=**

Specifies the address of the STOKEN of the space to be posted. The STOKEN is verified as part of posting process and if the address space no longer exists, a bad return code is passed to the caller. STOKEN= is Mutually exclusive with TTOKEN=.

**TTOKEN=**

Specifies the address of the TTOKEN of the task to be posted. If the address space associated with the TTOKEN no longer exists, a bad return code is passed to the caller. If the task no longer exists, the error routine is called. TTOKEN= is Mutually exclusive with STOKEN=.

**ERROR\_ROUTINE=**

Specifies a routine in common storage that gets control when the ECB cannot be posted because the task associated with the TTOKEN no longer exists or the SRB that was scheduled for the post has been purged. Routine might be running in SRB mode.

**ERROR\_PARM=**

Specifies a parameter to pass to the error routine (4 bytes). Pass in high half of R1 to the error routine.

**IRB=YES|NO**

Specifies whether the POST should be performed directly from the SRB (IRB=NO) or by an IRB attached to the job step TCB or TMP TCB (IRB=YES). TTOKEN=addrx must be provided. NO is the default.

**Operands common to all caller types:****COMP=**

Specifies completion code that should be supplied on the post.

**ERRET=**

Specifies a label to be branched to or a register to be branched on if a nonzero return code is returned in R15 (for example, the target address space is not available). This parameter is optional.

**OKRET=**

Specifies a label to be branched to or a register to be branched on if a zero return code is returned in R15. This parameter is optional.

**Environment**

- All environments.
- \$WAIT cannot occur.

**\$XMPOST**

---

## Appendix A. Using JES2 table pairs

---

### What are JES2 table pairs?

Table pairs provide a facility to modify, delete, or add JES2 processing and/or function. Changes made to JES2 processing using table pairs are generally less prone to error than are changes made through installation exits because JES2 macros generate the tables and generally requires you to write less executable code.

The term *table pair* is actually a misnomer; it is a representation of three pointers to the following three distinct sets of tables:

- **JES2 tables** - These are the tables defined by JES2, which are shipped by IBM, and provide the default processing specifications. There is (at most) one set of JES2 tables per table pair.
- **Dynamic tables** - These tables are defined by installations or vendor products and are automatically associated with the table pair when the module in which they reside is loaded (using the `LOAD(xxxxxxxx)` initialization statement). They are used to extend, modify, add to, or delete the default processing specifications, and in most cases override the processing specified in the JES2 tables. The number of dynamic tables associated with a table pair is unrestricted.
- **User tables** - These tables are defined by installations or vendor products. They are used to extend, modify, add to, or delete default processing, and in most cases override the processing specified in the JES2 and dynamic tables. There is (at most) one set of user tables per table pair.

To use the table pairs, you must provide user or dynamic tables to be associated with a particular table pair. You can also create new table pairs (using the `$PAIR` macro), but this requires that you either link-edit them with JES2 modules or define the table addresses to JES2. Dependent on the tables you choose, using table pairs generally takes less detailed knowledge of JES2 code, function, and control block structure and content than does the writing of an exit.

Table pairs do not replace the need for exits. Table pairs and exits can provide added capability either independently or in conjunction with one-another.

---

### JES2 table pairs versus JES2 exits

When you code exit points you may be modifying JES2 processing or function, adding installation processing or function, or deleting some JES2 processing or function. The services available, and the environment where the exit is called all affect what you are capable of achieving at a particular exit point. Therefore, there may be an exit point where you are capable of modifying JES2 processing but where you are not capable of deleting JES2 function or adding installation function.

To use the exit facility, you must write exit modules to contain your exit routines. Your modules can be link-edited with JES2 (in certain instances) or they can be independent of JES2. (Best general practice is to keep exits separate from JES2 modules and then use the `LOAD` initialization statement to define them to JES2.) Coding exits requires detailed knowledge of JES2, its coding conventions, its functions, capabilities, and its control blocks both in content as well as structure.

When you code table pairs, you can modify JES2 processing or function, delete JES2 processing or function, or add installation processing or function. Unlike exit points, you can modify, delete, or add function without restriction. However, IBM does not recommend deleting JES2 function.

To use table pairs, you must create installation table pairs and possibly also supporting routines, then either link-edit them with JES2 modules or define the table addresses to JES2. If you wish to add an initialization statement to JES2, this generally requires nothing more than a table entry to define the statement and specification for where to place the input. If you require more specialized processing than that supplied by JES2, then you can create supporting routines. Few of JES2's initialization and command tables require supporting prescan or postscan supporting routines.

Table pairs provide a structured mechanism to change JES2 processing that imposes fewer constraints and less complexity than using exit points.

## Concepts

Table pairs in JES2 begin with a router control block that contains the table pair. The first address points to an installation table, the second address points to a JES2 table, and the third address is an anchor for a chain of dynamic tables.

Figure 2 on page 426 shows a table pair that is associated with the following tables:

- An IBM-supplied table that describes the elements 'TWO', 'FIVE', and 'SIX'.
- An installation table that describes the elements 'ONE', 'TWO', and 'THREE'.
- A dynamic table describing the elements 'THREE' and 'FOUR'.
- A dynamic table describing the element 'FIVE'.

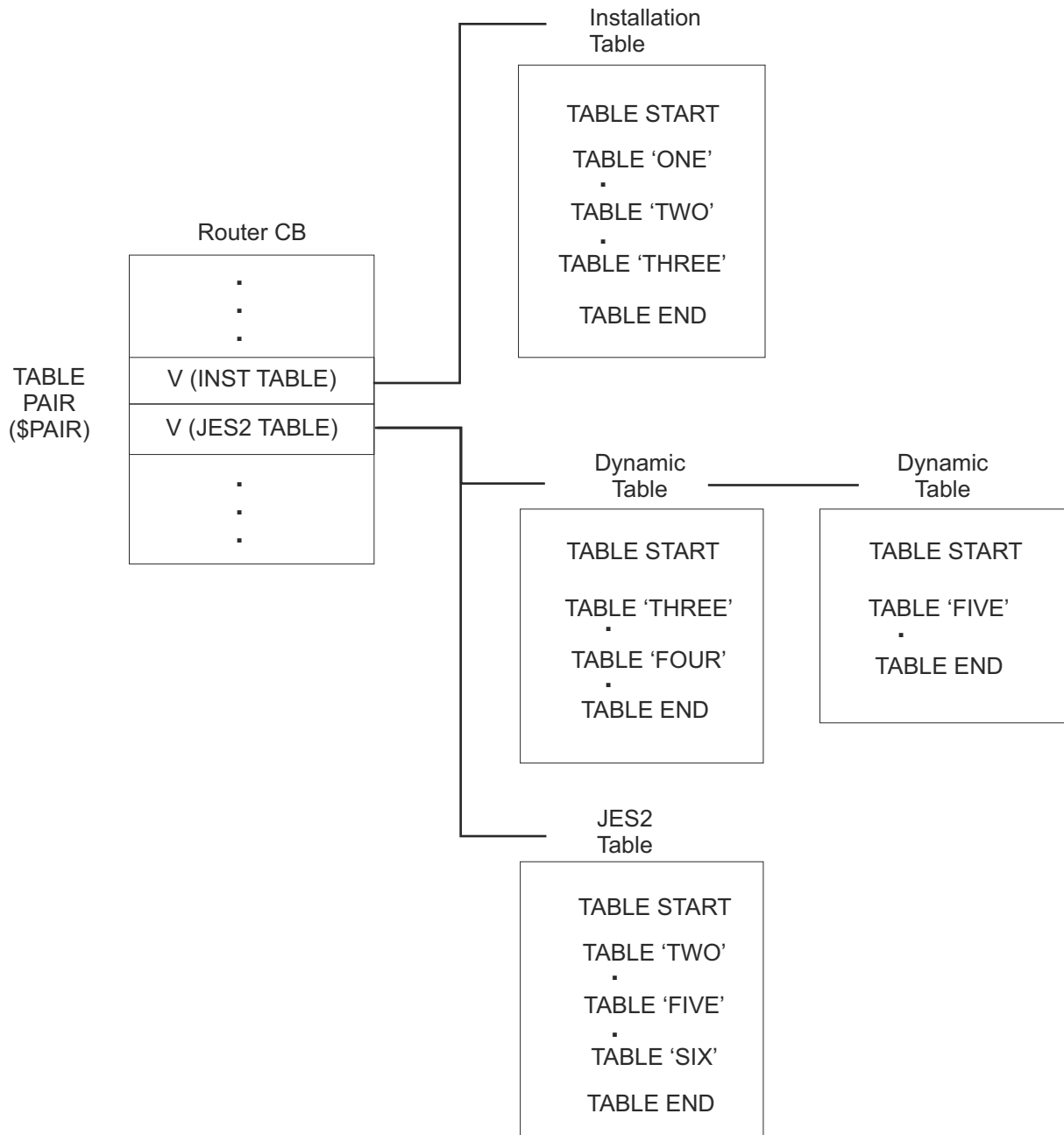


Figure 2. Table Pairs: A Diagrammatic View



JES2 uses these tables when it is processing the items 'ONE', 'TWO', 'THREE', 'FOUR', 'FIVE', and 'SIX'.

1. First, JES2 isolates the item to process (for example, 'ONE', 'TWO', or 'THREE') in the input source data.
2. Next, JES2 goes to the router control block to find the table pair to use to process the isolated item.
3. Then JES2 attempts to find the installation table. If the first table pair pointer is non-zero, then JES2 assumes this value is the address of the installation table. In this way, the installation table, if it exists, is **always** searched **prior** to the JES2 table. Initially all installation table pair pointers are set to zero. The installation table is optional and does not exist unless you create it.

If the item to process is located in the installation table, then processing continues using the installation table entry.

4. If the item is not found in the installation table, then JES2 searches the dynamic tables. If the item to process is located in a dynamic table, then processing continues using the dynamic table entry.
5. If the item to process was not found in either the installation table or the dynamic tables, then JES2 searches the JES2 table. If the item to process is found in the JES2 table, then processing continues using the JES2 table entry. If the item is not found in the JES2 table, then JES2 issues an error message.

Therefore, using the table arranged as described in [Figure 2 on page 426](#), the input items 'ONE', 'TWO', 'THREE', 'FOUR', 'FIVE', and 'SIX' are all processed. Assume they are encountered in that order.

1. First, the input item 'ONE' is processed. The item 'ONE' is located and isolated in the input stream. Next, the address of the installation table is found from the table pair in the router control block. The installation table is searched by examining each table element for a match for the input 'ONE'. In this example, the first table element matches the input. This table element is used by JES2 to process the input 'ONE'. Notice that the JES2 table does not include a table element that describes 'ONE'. Therefore, the installation has added some processing or function to JES2 without modifying any JES2 code.
2. Next, the input item 'TWO' is processed. The item 'TWO' is located and isolated from the data handed to JES2. Next the address of the installation table is found from the table pair in the router control block. The installation table is searched by examining each table element for a match for the input 'TWO'. In this example, the table element that matches the input is found later in the installation table. This table element is used by JES2 to process the input 'TWO'. Notice that the JES2 table includes a table element that describes 'TWO'. Because a match was found in the installation table, JES2 never searched the JES2 table. Thus, the installation has replaced or modified some processing or function without modifying any JES2 code.
3. Next, the input item 'THREE' is processed. Again, the table entry for 'THREE' is found in the installation table. Notice that one of the dynamic tables also includes a table element for 'THREE'. Because a match was found in the installation table, JES2 never searched the dynamic tables. Thus, the installation has replaced or modified some processing or function provided in those dynamic tables (which may have been provided, for example, by a vendor product) without actually modifying those tables.
4. Next, the input item 'FOUR' is processed. The item 'FOUR' is located and isolated from the data handed to JES2. Next the address of the installation table is found from the table pair in the router control block. The installation table is searched by examining each table element for a match on the input 'FOUR'. In this example, there is no match in the installation tables, so processing continues by searching the dynamic tables for an element that matches 'FOUR'. An element matching 'FOUR' is found in the dynamic tables, so that element is used by JES2 to process the input 'FOUR'. In this case there is no entry in the JES2 table matching 'FOUR', so this example represents function added by a dynamic table.
5. Next, the input item 'FIVE' is processed. The item 'FIVE' is located and isolated from the data handed to JES2. Next the address of the installation table is found from the table pair in the router control block. The installation table is searched by examining each table element for a match on the input 'FIVE'. In this example, there is no match in the installation tables, so processing continues by searching the dynamic tables for an element that matches 'FIVE'. An element matching 'FIVE' is found in the dynamic tables, so that element is used by JES2 to process the input 'FIVE'. In this case there is

an entry in the JES2 table matching 'FIVE', therefore, this example represents function that has been replaced through use of a dynamic table.

6. Finally, the input item 'SIX' is processed. The item 'SIX' is located and isolated from the data handed to JES2. Next the address of the installation table is found from the table pair in the router control block. The installation table is searched by examining each table element for a match for the input 'SIX'. In this example, there is no element that matches the input of 'SIX' in the installation table. Therefore, processing continues by searching the dynamic tables for an element that matches 'SIX'. When no matching element is found in the dynamic tables, the search continues in the JES2 table. When this table element is found in the JES2 table, the input 'SIX' is processed by this table element.

Deleting a table element is done by providing a null installation table that matches the JES2 table but provides no function.



**Attention:** IBM suggests that you do not delete JES2 tables.

If the installation table pointer is zero then no installation table exists, and JES2 uses the JES2 table to attempt to process the input. If the JES2 table pointer is zero, then the input must be found in the installation table or else the input is marked as incorrect.

The router control block contains one or more table pair addresses. The installation table fields of the table pair are defined as weak external V-type address constants. Therefore, installation tables can be link-edited with JES2 to have the linkage editor resolve the installation table addresses. If the installation table is not link-edited with JES2 then you must fill in the address of its table into the first of the correct table pairs.

The JES2 table entries are defined as V-type address constants. The linkage editor places the JES2 table addresses into the table pairs.

The dynamic table entries do not point to the dynamic tables directly. You should not modify the dynamic table entry yourself. The dynamic table entry is automatically established when a module containing a dynamic table is loaded through the `LOAD(xxxxxxxx)` initialization statement. Two separate dynamic tables override each other based on the order of `LOAD(xxxxxxxx)` statements. The first `LOAD` takes precedence over the second, and so on.

You should attempt to isolate as many of your installation-specific modifications as possible within user modules. JES2 provides user fields in a number of the commonly modified control blocks. For example, the UCT (user communication table) is effectively an extension of the HCT (HASP communication table). Other extension points are the tables that define processor control elements (PCEs), daughter task elements (DTEs), trace ID tables (TIDTABs), and the scan facility. These extensions should not reside within JES2 inline code, but rather in a user module or dynamic storage area accessed through installation exits. By using installation exits and JES2 macros, you can build fields to point to your own user tables to override the default JES2 tables. This eliminates the need to directly modify JES2 control blocks or copy JES2 code into user modules.

The JES2 table provides the default processing specifications. If you do not extend this table, JES2 will remain unmodified. However, if you choose to fill in the *user table* of the table pair you will be adding new function or overriding those JES2 specifications with what you have provided in the user table. For example, JES2 has specified that the minimum length of the RANGE parameter on the `PRT(nnnn)` initialization statement must be 5 (that is, all five characters must be coded). You can change this requirement by overriding the JES2 table by coding your own RANGE table entry. If you prefer the minimum number of characters to be 3, you could then code either RAN, RANG, or RANGE when specifying this parameter.

## Master control table

The master control table (\$MCT) contains all of the table pairs in JES2. The \$MCT contains the table pair pointers for:

- Processor creation (PCEs)
- Subtask creation (DTEs)

- Device definitions (DCTs)
- Trace identifiers
- Initialization options (for example, COLD, NOREQ, WARM, and so on)
- Main parameter statements (for example, CKPTDEF, SPOOLDEF, and so on)
- Operator commands (for example, \$D CKPTDEF, \$T SPOOLDEF, and so on)
- Work selection options.
- Block Extension Reuse Tables (BERTs)

The master control table (\$MCT) is pointed to from the \$HCT field \$MCT. Addresses of the installation tables can be resolved by either link-editing the installation table with JES2 or by placing the address of the installation table into the \$MCT through an exit. You can use Exit 0 for this purpose. Addresses of dynamic tables are resolved through the LOAD(xxxxxxxx) initialization statement. When the module containing the dynamic table is loaded, the dynamic table is linked to the appropriate table pair.

## General table coding conventions

All tables that you may build already have a JES2 counterpart available as an example in the JES2 code. The table type is defined by the corresponding macro, that is, use \$PCETAB to build a PCE table, \$SCANTAB to build a scan table. Each JES2 table begins with a TABLE=HASP specification. To code a user table, begin with a \$xxxTAB TABLE=USER specification (where xxx can be defined as SCAN, PCE, DCT, DTE, TID, BERT, and WS). To code a dynamic table, begin with a \$xxxTAB TABLE=DYNAMIC specification. Subsequent lines are added to specify the statement, command, or processor that you are defining, such as the PCE name and the module containing the processor's code for a PCE table entry or the valid parameter length and range in a SCAN table entry. Each table is then ended by coding a \$xxxTAB TABLE=END statement.

## Dynamic tables versus installation tables

As previously stated, JES2 may be extended through either the creation of a *user table* or a *dynamic table*. Which type of table to use for a specific extension depends on your needs. The following is a comparison of the advantages and disadvantages of each type.

### • User Table

- Only a single user table is allowed.
- The user table is linked to the table pair by one of the following:
  - The linkage editor, by naming the table USERxxxT and link-editing the table with the HASJES20 load module.
  - An installation exit (such as exit 0), which stores the address of the table in the field MCTxxxTU.
- The user table overrides the JES2 table and all dynamic tables.

### • Dynamic Table

- An unlimited number of dynamic tables may be provided.
- The dynamic table is linked to the table pair automatically when the load module containing it is loaded through the LOAD(xxxxxxxx) initialization statement.
- The dynamic table overrides the JES2 table but may be overridden by the user table.

IBM suggests that vendor products use dynamic tables for the following reasons:

- Your installation is not required to take any extra action to include the tables, such as merging them with your own or other vendors' tables.
- The dynamic tables can be easily overridden by a user table.

You can use either dynamic or user tables if you code your own tables. If you want to create separate sets of tables for different functions, you can use dynamic tables. If you want to override a dynamic table provided by a vendor, you can use a user table.

## Examples of table pairs

---

The remainder of this appendix is a series of examples of table pairs. The examples provide:

- The purpose of the table or function.
- A description of supporting control blocks and macros
- A description of what the table contains using a JES2 table element.
- Descriptions of the creation of installation tables and table element.

Appendix B, “Table pairs coding example,” on page 459 contains coded examples of the specific installation sample. These examples are interrelated to show how the tables can be used together. The examples show what you can do, not necessarily what you should do.

## Processor control elements (PCE) tables

---

The processor control elements tables can add installation processors (PCEs) to a JES2 system or override JES2 processors.

The JES2 PCE tables represent units of JES2 work. The JES2 dispatcher gives control to a PCE. No other PCE gains control until, and unless, this PCE directly relinquishes control. This is done when JES2 issues a \$WAIT. When a \$WAIT is done, control passes to the JES2 dispatcher, which saves the registers in the PCE control block that represents the JES2 processor and then dispatches another JES2 processor. A JES2 processor is ineligible for dispatching until it is \$POSTed.

PCEs can be generated during JES2 initialization or after initialization. Therefore, you can specify that a processor be created and be present for the life of JES2 or that it be created only upon installation demand (that is, after initialization).

You can also specify when the processor is given control. For example, you can specify if you want a processor to be given control concurrent with the HASPWARM processor for final initialization processing. Or, you can specify that the installation processor doesn't need to take control until initialization has completed but concurrent with the other JES2 processors. You can also indicate that a processor only gets control when it is \$POSTed for work.

Processors can also be associated with a device by pointing to a particular DCT table from the PCE table. This is a one-to-one correspondence, that is, one PCE is associated with one device.

## PCE control blocks and macros

The \$MCT table pair MCTPCETP points to JES2, installation, and dynamic PCE tables. The \$MCT field MCTPCETH points to the JES2 PCE table. The JES2 PCE table name is HASPPCET. The \$MCT field MCTPCETU points to the address of the installation table, if such a table exists.

The \$PCETAB macro builds both the JES2 and installation tables and table elements. This macro also contains the mapping macro for the PCE table and element. The \$PCEDYN macro invokes the \$PCEDYN service to provide JES2 a mechanism to dynamically attach and detach processors.

The \$GETABLE macro invokes the \$GETABLE service routine to obtain a table element from the JES2 table or the installation table. To obtain a PCE table, code TABLE=PCE operand. This macro returns the table element of the specified ID or, if LOOP is specified, return the next table element after the specified identifier.

The PCE control block contains fields that are required on a processor basis within the JES2 main task. The PCE is composed of a common section and an optional variable length section that is unique between processor types and contain processor specific information. The various processor types in JES2 include: input, JCL conversion, execution, output, print, and purge.

Register 13 in the JES2 main task points to the PCE common section which includes an OS-style save area at the top. In the PCE control block there are two installation-reserved fields, PCEUSER0 and PCEUSER1, in the common section.

Figure 3 on page 431 illustrates the contents of a PCE. The common area contains the OS-style save area at the top, followed by those fields that are common for all types of processors.

The variable length extension area is an optional extension to the common area that contains PCE-type specific information. Thus, the PCE extension for the reader PCE would be the same as other reader PCEs but different from the printer PCE extension area. The size of this extension area is specified on the PCE table.

er

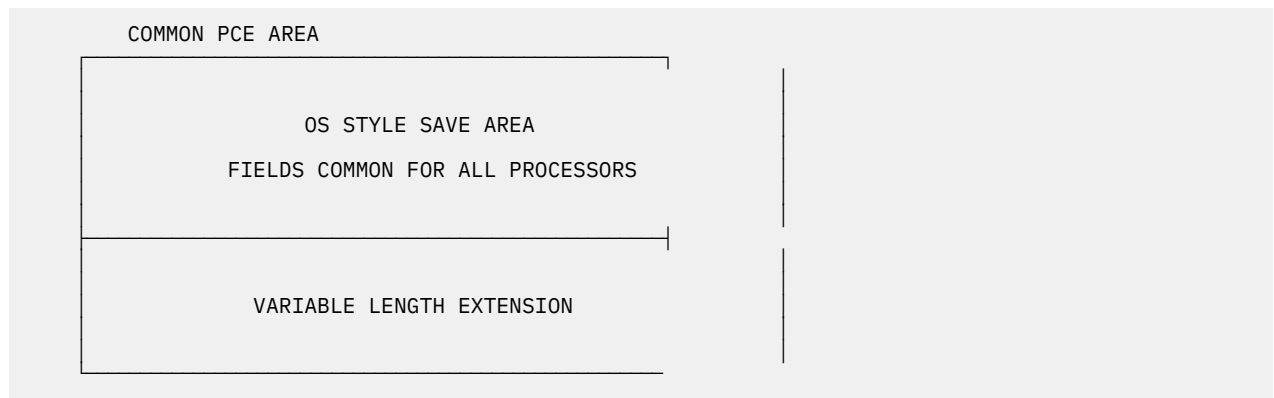


Figure 3. Common PCE Area Structure

JES2 processors maintain control of JES2 processing until they issue a \$WAIT macro. When the \$WAIT macro is issued, the JES2 dispatcher receives control and places the PCE on a queue for the requested resource. In JES2, the total number of resource queues is defined in \$HASPEQU through the equate named \$DRTOTAL. \$DRTOTAL is defined for 64 resource queue chains. When the processor issues a \$WAIT macro with a 1- to 5-character resource name, the macro and dispatcher place the processor on that \$DRxxxxx queue, where \$DRxxxxx is one of up to 64 resource names defined with an equate. JES2 resources start at 0 and increase; installation resource queues start at 63 and decrease.

Therefore, if a processor issued a '\$WAIT SCTY', the dispatcher would place the processor on the wait queue defined as \$DRSCTY. 'SCTY' is an installation resource. This processor remains on this queue until a \$POST SCTY is done. When the \$POST is done, the processors on the \$DRSCTY wait queue are put on the JES2 ready queue to be dispatched by the JES2 dispatcher.

All save areas in the JES2 main task are chained from a PCE. The PCE contains the PSV (PCE save area) that maps save areas chained from the PCE as well as the save area in the PCE itself. JES2 \$SAVE and \$RETURN services manage save areas chained off the PCE. The JES2 dispatcher uses the PCE save area for MVS service calls to save current register contents when the processor is \$WAITed.

To run the JES2 save areas, use field PCELPSV which points to the last (most recent) save area chained from the PCE and use PSVPREV in that save area to point to the previous save area. Do not use PSVNEXT from the PCE because MVS services, the JES2 dispatcher, or HASPSSSM may overlay this field.

JES2 save areas are similar to standard OS save areas in format, but not in the way they are used and accessed. Be aware that:

- Register 13 does not point to an available save area in the JES2 main task. You can do a STM into register 13, but the correct approach would be to do a \$SAVE to obtain a JES2 save area and save the registers in the JES2 main task environment.
- You cannot use register 13 to follow the chain of save areas from the JES2 main task, because register 13 is kept as an available save area for calls to MVS services, not JES2 routines.
- The save area format is different in that there are extra words on the end of JES2 save areas that JES2 uses to point to the PCE (PSVPCE) and the \$SAVE identifier at the location where the \$SAVE was issued (PSVLABAD).

Figure 4 on page 432 (Part 1) illustrates the chaining for JES2 save areas. The PCE field PCELPSV points to the last (most recent) JES2 save area. By using PSVPREV, the save areas can be chained back to the PCE. The save area in the PCE is available for use by other services that require OS-style save areas.

You can use the PCE field PSVPCE from any JES2 save area to obtain the PCE address as illustrated in Figure 4 on page 432 (Part 2). While running the JES2 save areas, the PSVNEXT field is valid. However, do not use this field from the PCE; it may not be valid.

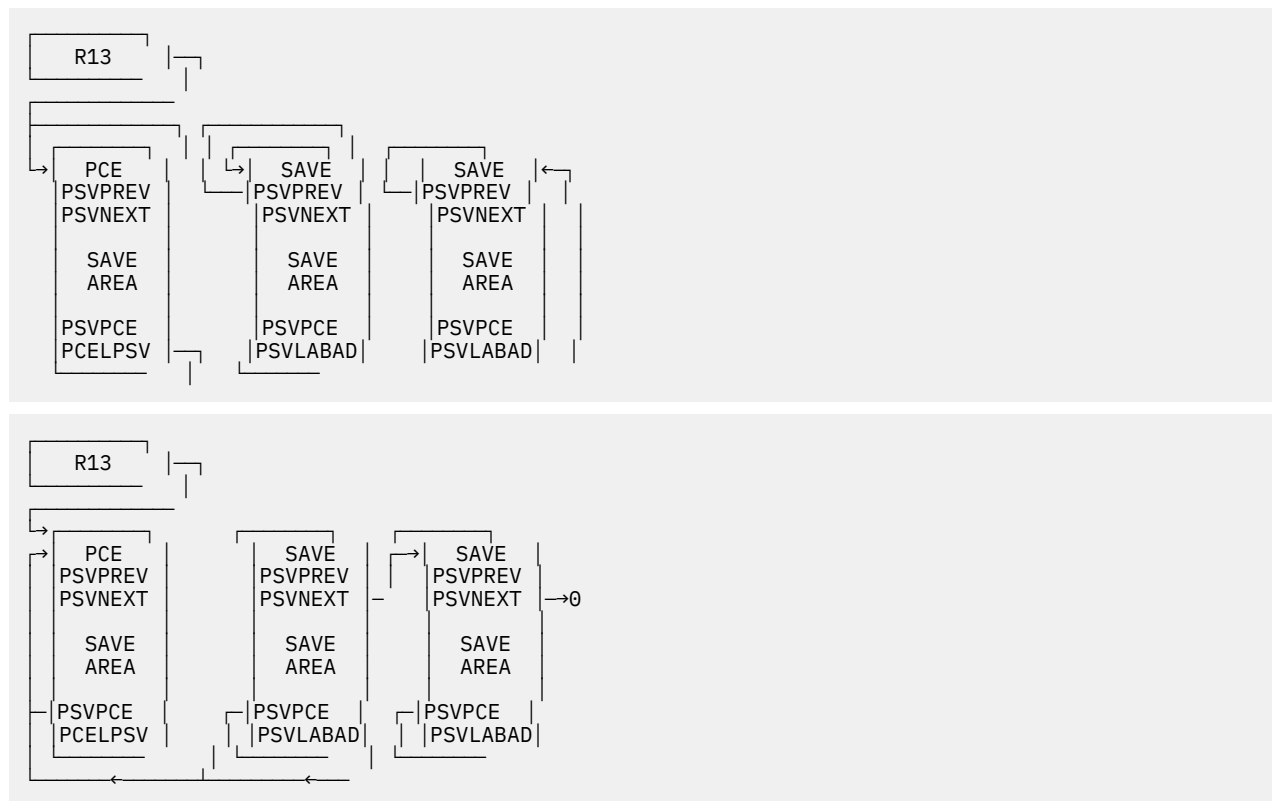


Figure 4. PCE Tables - Save Area Chaining

## A JES2 PCE table

Figure 5 on page 432 presents the JES2 PCE table. The table name is HASPPCET; the same as that specified in the V-type address constant in the \$MCT. The table is delimited by TABLE=HASP (to start of the table) and TABLE=END (to end of the table). The table element shown represents all the information that JES2 needs to define a JES2 reader processor. This table element is passed to the \$PCEDYN service to create the reader PCE.

Specifying whether it is a JES2 or an installation table determines default values for the ENTRYPT and CHAIN \$PCETAB operands. When \$PCETAB is specified with operands other than TABLE=, the macro generates a table element. All JES2 PCEs are defined within this single table.

```
HASPPCET $PCETAB TABLE=HASP
          $PCETAB NAME=...
RDRPCET  $PCETAB NAME=RDR,
          DESC='READER'
          DCTTAB=RDRDCTT,
          MODULE=HASPRDR,
          ENTRYPT=MAPRDRA,
          CHAIN=$RDRPCE,
          COUNTS=$NUMRDRS,
          MACRO=$RDRWORK,
          WORKLEN=RDWLEN,
          GEN=INIT,
          DISPTCH=WARM,
          PCEFLGS=0,
          FSS=NO,
          PCEID=(PCELCLID,PCERDRID)
          $PCETAB NAME=...
          $PCETAB TABLE=END
```

Figure 5. The JES2 PCE Table

# An installation PCE table

Figure 6 on page 433 illustrates an installation PCE.

```
USERPCET $PCETAB TABLE=USER
SCTYPCET $PCETAB NAME=SCTY,
          DESC='SECURITY'                                X
          DCTTAB=*-*,                                    X
          MODULE=HASPXJ00,                                X
          ENTRYPT=UCTMSCTY,                              X
          CHAIN=UCTSYPCE,                                X
          COUNTS=UCTSYNUM,                              X
          MACRO=$SCYWORK,                                X
          WORKLEN=SCYLEN,                                X
          GEN=INIT,                                       X
          DISPTCH=WARM,                                  X
          PCEFLGS=0,                                     X
          FSS=NO,                                         X
          PCEID=(0,UPCESCTY)
$PCETAB TABLE=END
```

Figure 6. Example of an Installation PCE Table

# A dynamic PCE table

Figure 7 on page 433 illustrates an alternative method of defining the installation table in Figure 6 on page 433 through use of a dynamic table.

```
MYPCETAB $PCETAB TABLE=DYNAMIC
SCTYPCET $PCETAB NAME=SCTY,
          DESC='SECURITY'                                X
          DCTTAB=*-*,                                    X
          MODULE=HASPXJ00,                                X
          ENTRYPT=UCTMSCTY,                              X
          CHAIN=UCTSYPCE,                                X
          COUNTS=UCTSYNUM,                              X
          MACRO=$SCYWORK,                                X
          WORKLEN=SCYLEN,                                X
          GEN=INIT,                                       X
          DISPTCH=WARM,                                  X
          PCEFLGS=0,                                     X
          FSS=NO,                                         X
          PCEID=(0,UPCESCTY)
$PCETAB TABLE=END
```

Figure 7. Example of a Dynamic PCE Table

To create the installation PCE table illustrated in Figure 6 on page 433 or Figure 7 on page 433, you need to code the following operands on the \$PCETAB macro:

Operand	Description
NAME=SCTY	The name of the PCE
DESC=SECURITY	The description of the processor. The word ‘PROCESSOR’ is appended to the end of the value specified on the DESC operand.
DCTTAB=*-*	An indicator that there was no DCT table and that the processor is not associated with a device
MODULE=HASPXJ00	The name of the module to contain the processor code
ENTRYPT=UCTMSCTY	The field to hold the entry point address. The \$UCT field UCTMSCTY holds the address of the routine USCTPCE
CHAIN=UCTSYPCE	The name of the \$UCT field to hold the pointer to the first security PCE

Operand	Description
COUNTS=UCTSYNUM	Where the \$PCEDYN service routine is to find out how many PCEs of this type it can create and to keep track of how many it has created.
MACRO=\$SCYWORK	The PCE's own variable extension area The \$SCYWORK macro maps this extension area.
WORKLEN=SCYLEN	The length of the variable extension area. SCYLEN is an equate in the \$SCYWORK macro.
GEN=INIT	The indicator to generate the processor during initialization.
DISPTCH=WARM	The indicator that the processor should receive control after warm start processing. This assumes that the security processor is not be needed during warm start processing.
PCEFLGS=0	<p>The indicator that the PCE has no special requirements. The PCEFLGS operand specifies the initial value the PCE PCEFLAGS field contains after it is created by \$PCEDYN. If the initial state of the processor should be that it:</p> <ul style="list-style-type: none"> <li>• Should be traced, specify PCETRACE.</li> <li>• Should be marked as permanently exempt from non-dispatchability, specify PCEDSPXP. If the processor should never be marked non-dispatchable, then set this value.</li> <li>• Should be marked as temporarily exempt from non-dispatchability, specify PCEDSPXT. This value would be specified if some processing must be completed by this processor that would fail if the processor was marked non-dispatchable.</li> <li>• Cannot wait in the case of an I/O error, then specify PCENWIOP.</li> </ul>
FSS=NO	The indicator that the processor should not run in FSS mode.



Operand	Description
PCEID=(0,UPCESCTY)	<p>The indicator that this is a processor that is not associated with a device. The identifier of the processor is 255. Installation-specified identifiers should start at 255 and decrease. JES2 processors start at 1 and increase. Code an equate in the \$UCT named UPCESCTY and set it to 255. The PCEID operand specifies the type and identifier of the processor as follows:</p> <p><b>Processor Type Meaning</b></p> <p><b>0</b> Non-device processor</p> <p><b>PCELCLID</b> Local special PCE identifier</p> <p><b>PCERJEID</b> Remote special PCE identifier</p> <p><b>PCENJEID</b> Network special PCE id, indicates NJE or XFE JT/JR/ST/SR</p> <p><b>PCEINRID</b> Initial special PCE identifier</p> <p><b>PCEPRSID</b> Printer special PCE identifier</p> <p><b>PCEPUSID</b> Punch special PCE identifier</p> <p><b>PCEXFRID</b> XFR special PCE identifier</p>

## Coding the other pieces

In addition to coding the installation PCE table, you need to:

- Write a HASPXJ00 module to hold the PCE code
- Create a macro called \$SCYWORK to map the PCE extension. \$SCYWORK must contain a field named SCYLEN to define the length of the extension area.
- Code these fields in the installation \$UCT:
  - UCTMSCTY DC A(\*-\*) ADDR OF ENTRYPT  
The address of the entry point for the HASPXJ00 module for the installation PCE is held in the UCTMSCTY field.
  - UCTSYPCE DC A(\*-\*) ADDR OF SCTY PCE  
The address of the first security PCE is chained from the UCTSYPCE field.
  - UCTSYNUM DC H'1',H'0'  
A two halfword field where the first field defines the number of security PCEs that are to be created and the second indicates to \$PCEDYN how many have been created.
  - UPCESCTY EQU 255 ID OF SCTY PCE  
The identifier of the PCE (set at 255)
  - \$DRSCTY EQU 63 DISP SEC RESOURCE  
A dispatching security resource that tells the PCE that some work is ready for it to process The installation PCE will '\$WAIT SCTY' (which will result in the PCE being put on the resource queue of

63) for work. When there is work for it to do, it is \$POSTed for SCTY (that is, \$DRSCTY = 63) and put on the ready queue.

- Code Exit 0

The Exit 0 code is required to do three things.

- Obtain the \$UCT and place the \$UCTs address in the \$UCT field in the \$HCT, or better still, into a name/token using the \$TOKENSR service.
- Initialize the \$UCT fields. The fields that must be initialized include, at least, the UCTMSCTY, UCTSYPCE, and the first halfword of UCTSYNUM.
- Place the installation PCE table address in the MCTPCETU field in the \$MCT in module HASPTABS. This is not necessary for dynamic tables. Dynamic tables should be linked to the table pair by placing a LOAD initialization statement in your JES2 initialization stream for the module containing the dynamic PCE table.

## Daughter task element (DTE) tables

---

The daughter task element tables represent subtasks in JES2. In JES2 subtasks do work that may require MVS WAITs. MVS WAITs are not tolerated in the JES2 main task.

The DTEs are tabular in the \$DTETABs. This provides the capability to add installation-defined subtasks and to override JES2 subtasks. We do not recommend that you delete JES2 subtasks. The tables that define the JES2 subtasks reside in the module, HASPTABS.

The DTE is available to the main task (a PCE processor) and the subtask and assists communication between the two environments.

To serialize the communications between the main task and the subtask, follow MVS dispatching methods. This involves the use of \$WAITs and MVS POSTs from the main task and MVS WAITs and POSTs from the subtask. Never issue an MVS WAIT from the JES2 main task and never issue a JES2 \$WAIT from a JES2 subtask.

## DTE control blocks and macros

The \$MCT table pair MCTDTETH points to JES2, installation, and dynamic DTE tables. The \$MCT field MCTDTETH points to the JES2 DTE table. The JES2 table name is HASPDTET. The \$MCT field MCTDTETU contains the address of the installation table, if such a table exists.

The \$DTETAB macro builds both the JES2 and installation tables and table elements. This macro also contains the mapping macro for the DTE table and element. The \$DTEDYN macro using the \$DTEDYN service provides JES2 a mechanism to dynamically attach and detach subtasks.

The \$GETABLE macro invokes the \$GETABLE service routine to obtain a table element from JES2 table or installation table. To obtain a DTE table, code the TABLE=DTE operand. This macro returns the table element of the specified ID or, if LOOP is specified, it will return the next table element after the specified ID.

The DTE contains fields that are required on a subtask basis within the JES2 subtasks. The DTE is composed of a common section and an optional variable length section that is unique between subtask types and contain subtask-specific information. The subtask names are: HASPIMAG, HOSALLOC, HOSPOOL, HASPACCT, HASPVTAM, HASPWTO, HOSCNVT, HASPOFF, HASPCKCF, and HASPCKVR.

Register 13 in the JES2 subtask points to the DTE which is an available save area.

The following four fields are used for subtask recovery:

- \$STABNDA - this field in the \$HCT contains the address of the general subtask recovery routine. If you code an ESTAE (highly recommended), use this routine as the recovery routine. This recovery routine takes three "exit" calls, depending upon whether the following three fields are nonzero.
- DTEVRXAD - this field in the DTE contains the address of a VRA "exit" routine which receives control from the JES2 general subtask recovery routine to complete the variable recording area (VRA) in the SDWA. In this way, subtask-specific data is saved.

- DTERTXAD - this field in the DTE contains the address of a retry routine which receives control to attempt to retry. The general JES2 recovery routine issues a SETRP to a general retry routine. This general retry routine then gives control to the specified retry routine for this subtask. The subtask retry routine should issue a \$SETRP to a resumption point or percolate. If the subtask is to retry or percolate, the retry routine should prepare for the event.
- DTESXAD - this field in the DTE contains the address of a clean-up routine which receives control from the JES2 general subtask recovery routine to attempt subtask-specific clean-up. There are two valid return codes from this recovery routine:
  - 0 - continue normal recovery, clean-up successful
  - 4 - unrecoverable subtask error, abend JES2 main task a CALLRTM.

There are pointers in the \$HCT for the JES2 subtasks (DTEs) for each type of subtask. The chain heads are:

- 0 - no subtasks for this type exist
- \$DTEIMAG - points to the image subtasks
- \$DTEALOC - points to the allocation subtask
- \$DTESPOL - points to the spool subtasks
- \$DTESMF - points to the SMF subtask
- \$DTEVTM - points to the VTAM subtask
- \$DTEWTO - points to the WTO subtask
- \$DTECNVT - points to the converter subtasks
- \$DTEOFF - points to the offload subtasks
- \$DTESTID - this field contains the subtask identifier.
- \$DTECKCF - points to the checkpoint on CF subtask
- \$DTECKVR - points to the checkpoint versions subtask

## A JES2 DTE table

Figure 8 on page 437 illustrates the JES2 DTE table. The table name is HASPDTET; the same as that specified in the V-type address constant in the \$MCT. The table is delimited by a TABLE=HASP (to start the table) and a TABLE=END (to end the table). The table element shown represents all the information that JES2 needs to define a JES2 converter subtask. This is the table element that is passed to the \$DTEIDYN service to create the converter DTE. Whether it is a JES2 or an installation table determines some default values for the EPLOC and HEAD \$DTETAB operands. When you specify \$DTETAB with operands other than TABLE=, the macro generates a table element. When the TABLE=END is encountered, the table is closed.

```
HASPDTE $DTETAB TABLE=HASP
$DTETAB NAME=...
$DTETAB NAME=CONVERT,
          ID=DTEIDCNV,
          EPNAME=HOSCNVT,
          EPLOC=MAPCNVA,
          HEAD=$DTECNVT,
          WORKLEN=DCNVLEN,
          GEN=NO,
          STAE=NO,
          SZERO=NO
$DTETAB NAME=...
$DTETAB TABLE=END
```

Figure 8. The JES2 DTE Table

All JES2 subtasks are defined within this single table.

# An installation DTE table

Figure 9 on page 438 illustrates an installation DTE table.

USERDTET	\$DTETAB	TABLE=USER	
	\$DTETAB	NAME=SECURITY,	X
		ID=UDTESCTY,	X
		EPNAME=USCTDTE,	X
		EPLOC=UCTMDSCY,	X
		HEAD=UCTSYDTE,	X
		WORKLEN=SCDLEN,	X
		GEN=NO,	X
		STAE=NO,	X
		SZERO=YES	
	\$DTETAB	TABLE=END	

Figure 9. Example of An Installation DTE Table

# A dynamic DTE table

Figure 10 on page 438 illustrates an alternative method of defining the installation table in Figure 9 on page 438 through the use of a dynamic table.

MYDTETAB	\$DTETAB	TABLE=DYNAMIC	
	\$DTETAB	NAME=SECURITY,	
		ID=UDTESCTY,	X
		EPNAME=USCTDTE,	X
		EPLOC=UCTMDSCY,	X
		HEAD=UCTSYDTE,	X
		WORKLEN=SCDLEN,	X
		GEN=NO,	X
		STAE=NO,	X
		SZERO=YES	
	\$DTETAB	TABLE=END	

Figure 10. Example of a Dynamic DTE Table

To create the installation DTE table illustrated in Figure 9 on page 438 or Figure 10 on page 438, you need to code the following on the \$DTETAB macro:

Value	Description
NAME=SECURITY	The name of the subtask used in JES2 messages
ID=UDTESCTY	The identifier of the processor. Installation specified identifiers should start at 255 and decrease because JES2 subtask identifiers start at 0 and increase. There is an equate specified in the \$UCT named UDTESCTY set to 255.
EPNAME=USCTDTE	The name of the entry point to the subtask code in module HASPXJ00 JES2 uses USCTDTE on the MVS IDENTIFY call. The field that holds the entry point address, UCTMDSCY, is in the \$UCT. It will hold the address of the routine USCTDTE. Therefore, code
EPLOC=UCTMDSCY	The entry point address in the \$UCT. It contains the address of the routine USCTDTE.
HEAD=UCTSYDTE	The name of the chain field. The \$UCT field to hold the pointer to the first security subtask is UCTSYDTE.
WORKLEN=SCDLEN	The length of the variable extension area of the security subtask is defined through an equate called SCDLEN in macro \$SCDWORK.
GEN=NO	The indicator that the processor should not be generated automatically.

Value	Description
STAE=NO	The indicator that the subtask is not to be detached with the STAE operand specified on the MVS DETACH call specify
SZERO=YES	The indicator that the subtask shares subpool 0.

## Coding the other pieces

In addition to coding the installation DTE table, you need to:

- Write a HASPXJ00 module that holds the DTE subtask code
- Create a macro called \$SCDWORK to map the DTE extension. An equate named SCDLEN is required within the macro to define the length of the extension area needed.
- Code two fields and one equate in the installation \$UCT
  - UDTESCTY EQU 255 ID OF SCTY DTE  
An equate for the identifier of the subtask. We specify the equate UDTESCTY with a value of 255.
  - UCTMDSCY DC A(\*-\*) ADDR OF ENTRYPT  
The address of the entry point for the HASPXJ00 module for the installation DTE
  - UCTSYDTE DC A(\*-\*) ADDR OF SCTY DTE  
The address of the first security DTE
- Code Exit 0 code. Exit 0 must:
  - Obtain the \$UCT and place the \$UCT's address in the \$HCT.
  - Initialize the \$UCT.
  - Place the installation DTE table address in the MCTDTETU field in the \$MCT in module HASPTABS. This is not necessary for dynamic tables. Dynamic tables should be linked to the table pair by placing a LOAD initialization statement in your JES2 initialization stream for the module containing the dynamic DTE table.

## Work selection (WS) tables

The WS tables are used to add installation work selection criteria to a JES2 system or override JES2 work selection criteria in JES2.

## WS control blocks and macros

The \$MCT table pairs for work selection tables are MCTPRWTP for printers, MCTPUWTP for punches, MCTJTWTP for offload job transmitters, MCTJRWTP for offload job receivers, MCTSTWTP for offload sysout transmitters, MCTSRWTP for offload sysout receivers, MCTLJWTP for NJE line job transmitters, and MCTLSWTP for NJE line sysout transmitters. If you want to create a dynamic work selection table for one of these device types, you should code \$WSTAB TABLE=(DYNAMIC,*pair-offset*), where *pair-offset* is a valid table pair for a work selection table. You may list multiple table pairs if the work selection tables are to be associated with multiple device types.

The \$MCT fields for installation work selection tables are MCTPRWTU for printers, MCTPUWTU for punches, MCTJTWU for offload job transmitters, MCTJRWU for offload job receivers, MCTSTWU for offload sysout transmitters, MCTSRWU for offload sysout receivers, MCTLJWU for NJE line sysout transmitters, and MCTLSWU for NJE line sysout transmitters. If you want to link-edit an installation table with JES2 you must name your tables USERPRWT for printers, USERPUWT for punches, USERJTWU for offload job transmitters, USERJRWU for offload job receivers, USERSTWU for offload sysout transmitters, USERSRWU for offload sysout receivers, USERLJWU for NJE line job transmitters, and USERLSWU for NJE line SYSOUT transmitters. The installation table must then be link-edited with HASJES20. The JES2 WS tables are pointed to from the \$MCT using the MCT above and table names.

The \$WSTAB macro builds both the JES2 and installation tables and table elements. This macro also contains the mapping macro for the WS tables and elements.

## A JES2 WS table

Figure 11 on page 440 illustrates the JES2 work selection table. The table name is HASPPRWT; the same as that specified for the V-type address constant in the \$MCT. The table is delimited by TABLE=HASP (to start of the table) and TABLE=END (to end of the table). The table element shown represents the information that JES2 needs to define the JES2 criterion for JOBNAME. This is the table element that is passed to the \$#GET service routine which returns eligible JOEs for processing based upon the work selection list defined for the printer.

When \$WSTAB is specified with operands other than TABLE=, the macro generates a table element. All of the JES2 printer work selection criteria are defined within this single table.

```
HASPPRWT $WSTAB TABLE=HASP
          $WSTAB NAME=...
          $WSTAB NAME=JOBNAME,
                      MINLEN=3,
                      FLD=JQEJNAME,
                      CB=JQE,
                      DEVFLD=DCTJOBNM,
                      DEVCB=DCT,
                      RTN=COMPARE
          $WSTAB NAME=...
          $WSTAB TABLE=END
```

Figure 11. The JES2 WS Table

## An installation WS table

Figure 12 on page 440 describes an installation work selection criteria to select output that is beyond a specified limit for offload processing.

During periods of peak spool use (for example, end of month or end of year processing), you may be interested in using the spool offload facility to offload jobs that are using a large amount of JES2 spool. To achieve this, you would like there to be an additional work selection criterion on the offload SYSOUT transmitter. This operand would indicate at what spool usage threshold a job would be when it would be offloaded from the system.

```
USERSTWT $WSTAB TABLE=USER
          $WSTAB NAME=TRKGRP,
                      MINLEN=2,
                      ALIAS=TG,
                      FLD=JQETGNUM,
                      CB=JQE,
                      DEVFLD=DCTUSER0,
                      DEVCB=DCT,
                      RTN=WSTRKGRP
          $WSTAB TABLE=END
```

Figure 12. Example of an Installation WS Table

Coding the installation work selection table involves deciding what values you want to expose to your operators. For example, the work selection operand that is seen and entered by the operators is TRKGRP, which indicates that work is selected based on the number of track groups (spool space) that has been allocated to a job.

## A dynamic WS table

Figure 13 on page 441 illustrates an alternative method of defining a WS table from Figure 12 on page 440 through use of a dynamic table.

```

MYWSTAB $WSTAB TABLE=(DYNAMIC,MCTSTWTP)
        $WSTAB NAME=TRKGRP,
            MINLEN=2,
            ALIAS=TG,
            FLD=JQETGNUM,
            CB=JQE,
            DEVFLD=DCTUSER0,
            DEVCB=DCT,
            RTN=WSTRKGRP
        $WSTAB TABLE=END

```

Figure 13. Example of a Dynamic WS Table

Coding the installation work selection table involves deciding what values you want to expose to your operators. For example, the work selection operand that is seen and entered by the operators is TRKGRP, which indicates that work is selected based on the number of track groups (spool space) that has been allocated to a job.

The following is a description of the \$WSTAB operands used to create the table illustrated in [Figure 12 on page 440](#).

Operand	Description
NAME	The name of the individual work selection criterion.
MINLEN	The minimum length of the NAME operand.
ALIAS=TG	The accepted abbreviation for track groups is TG. To prevent confusion, specify an alias of TRKGRP that may make more sense to your operators.
FLD=JQETGNUM	<p>The field contains the number of track groups allocated to the job. This field determines whether there is a match with the device field. Therefore, the FLD operand is set to JQETGNUM. Thus, the job's number of track groups obtained from field JQETGNUM determines whether the offload SYSOUT transmitter "device" should select this job for transmitting.</p> <p>The field FLD=JQETGNUM is located in the control block JQE. The JQE (job queue element) is a control block that represents the job while it is in the system.</p> <p>So, the job's field JQETGNUM is compared against a threshold value set for the offload SYSOUT transmitter "device".</p>
DEVFLD=DCTUSER0	The threshold value for the transmitter device is in the field DCTUSER0. The DCTUSER0 field is set by the operator as the threshold value.
DEVCB=DCT	The device control block is DEVCB=DCT. The device field DCTUSER0 is located in the control block DCT (device control table). DCTs define devices to JES2. Thus, every device in JES2 has a DCT; this includes offload SYSOUT transmitters.
RTN=WSTRKGRP	discussed earlier, a work selection routine has to gain control to verify that the amount of spool space allocated to a job (JQETGNUM) is greater than the threshold specified by the user for the device (DCTUSER0). This is because while the job is in conversion or execution, JQETGNUM holds an offset into the checkpoint area which contains the number of track groups allocated to the job. The routine is WSTRKGRP. This routine must be link-edited with this table entry so that the routine's address can be resolved.

## Coding the other pieces

In addition to coding the installation work selection table, you need to

- Define the installation work selection routine (WSTRKGRP)
- Code Exit 0. Exit 0 code must:
  - Obtain the \$UCT and place the \$UCT's address in the \$HCT
  - Initialize the \$UCT
  - Place the installation Work Selection table address in the MCTSTWTU field in the \$MCT in module HASPTABS. This is not necessary for dynamic tables. Dynamic tables should be linked to the table pair by placing a LOAD initialization statement in your JES2 initialization stream for the module containing the dynamic WS table.

## Trace identifiers (TID) tables

---

Trace identifiers tables are used to add installation trace identifiers to a JES2 system or to override JES2 trace identifiers in JES2.

### TID control blocks and macros

The \$MCT table pair MCTTIDTP points to JES2, installation, and dynamic DTE tables. The \$MCT field MCTTIDTH points to the JES2 TID table. The JES2 TID table name is HASPTIDT. The \$MCT field MCTTIDTU contains the address of the installation table, if such a table exists.

The \$TIDTAB macro builds both the JES2 and installation tables and table elements. This macro also contains the mapping macro for the TID table and element.

The \$TRACE facility uses the TID tables to determine what identifiers are valid and what formatter routines receive control.

The \$TRACE executable macro allocates a JES2 trace table entry in an active trace table and returns its address. Optionally, \$TRACE initializes the trace table entry (TTE) based upon parameters passed. The JES2 event trace facility is called to perform the TTE allocation.

\$TRACE can be specified anywhere in the JES2 system (including the HASCnnnn user environment load modules) except in routines running as disabled interrupt exits (for example, an IOS appendage). Register 13 must point to a usable OS-style save area. You must also code the \$TRP macro on the \$MODULE statement to provide the required mapping.

The \$GETABLE macro provides access to the TID tables. The \$GETABLE macro invokes the \$GETABLE service routine to obtain a table element from the JES2 or installation table. To obtain a TID table, code the TABLE=TID operand. This macro returns the table element of the specified ID, or, if LOOP is specified, it returns the next table element after the specified ID.

The \$TLGWORK macro maps the event trace log processor variable extension area. This macro contains fields that are specific for the processor and needed by the installation format routines.

To use the trace interface, it is necessary to understand the structures of the primary control blocks. These control blocks include the trace table prefix (TTP) and the trace table entry (TTE). The TTP describes the entire trace table while the TTE describes elements within the trace table.



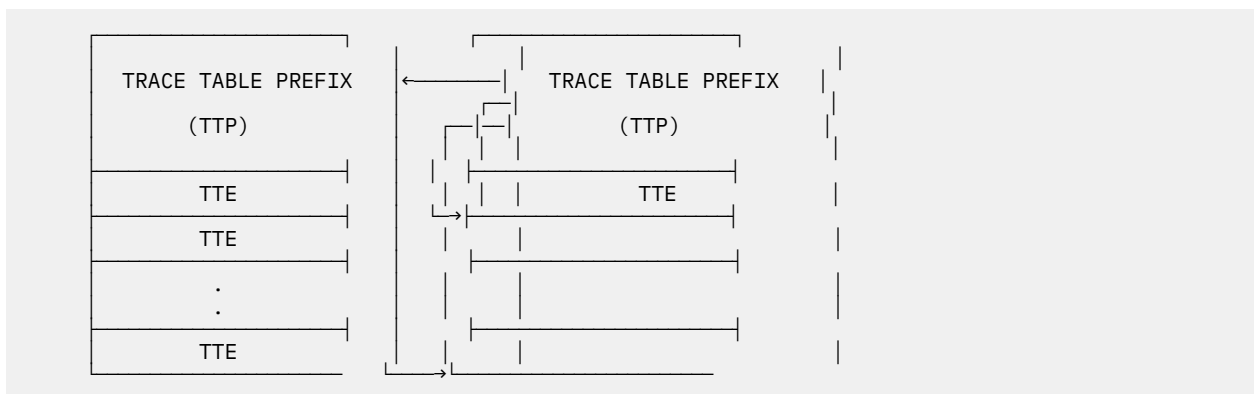


Figure 14. Trace Table Structure

Figure 14 on page 443 shows two trace tables containing trace table prefixes. The TTP has three pointers. The first pointer points to the previous trace table, the second pointer points to the end of the table, and the final pointer points to the next available spot in the trace table.

Trace tables are made up of as many TTEs as can fit in the trace table. The TTEs are not a fixed size, but are the size as specified on the \$TRACE macro call. The top of the TTE contains the fields mapped by the \$TTE macro that describe the data contained in the TTE.

## A JES2 TID table

Figure 15 on page 443 illustrates the JES2 TID table. The table name is HASPTIDT; the same as that specified in the V-type address constant in the \$MCT. The table is delimited by TABLE=HASP (to start of the table) and TABLE=END (to end of the table). The table element shown represents all the information that JES2 needs to define JES2 trace identifier 1 for the tracing of \$SAVES. This is the table element that is passed to the \$TRACE facility.

When \$TIDTAB is specified with operands other than TABLE=, the macro generates a table element. In Figure 15 on page 443, the table element that is generated is for trace identifier 1. All JES2 trace identifiers are defined within this single table.

```
HASPTIDT $TIDTAB TABLE=HASP
          $TIDTAB ID=...
          $TIDTAB ID=001,
              FORMAT=TROUT001,
              NAME=$SAVE
          $TIDTAB ID=...
          $TIDTAB TABLE=END
```

X  
X

Figure 15. The JES2 TID Table

## An installation TID table

Figure 16 on page 443 illustrates an installation trace table

```
USERTIDT $TIDTAB TABLE=USER
          $TIDTAB ID=255,
              FORMAT=TROUT255,
              NAME=SAFCALL
          $TIDTAB TABLE=END
```

X  
X

Figure 16. Example of an Installation TID Table

## A dynamic TID table

Figure 17 on page 444 illustrates an alternative method of defining a trace table from Figure 16 on page 443 through use of a dynamic table.

```

MYTIDTAB $TIDTAB TABLE=DYNAMIC
          $TIDTAB ID=255,
          FORMAT=TROUT255,
          NAME=SAFCALL
          $TIDTAB TABLE=END

```

Figure 17. Example of a Dynamic TID Table

To create the installation TID table illustrated in [Figure 16 on page 443](#) or [Figure 17 on page 444](#), you need to code the following on the \$TIDTAB macro.

Operand	Description
ID=255	The installation identifier. Installation identifiers should start at 255 and decrease,
FORMAT=TROUT255	The name of the format routine is TROUT255, for TRace OUTput for identifier 255.
NAME=SAFCALL	The name that is associated with the trace entry should be SAFCALL, because the function of this trace identifier is to trace a SAF call.

## Coding the other pieces

In addition to coding the installation TID table, you need to do the following:

- Code Exit 0. Exit 0 must:
  - Obtain the \$UCT and place the \$UCTs address in the \$HCT.
  - Initialize the \$UCT.
  - Place the installation TID table address in the MCTTIDTU field in the \$MCT in module HASPTABS. This is not necessary for dynamic tables. Dynamic tables should be linked to the table pair by placing a LOAD initialization statement in your JES2 initialization stream for the module containing the dynamic trace id table.
- Provide the format routine. The installation format routine cannot itself issue a TRACE=YES on its \$SAVE or \$RETURN. The registers upon entry to the format routine are as follows:
  - Register 1 - points to the TTP for the trace table that contains the entry as defined by the installation TIDTAB.
  - Register 2 - points to the TTE that contains the data that the installation \$TRACE macro saved. This is the data to be formatted by the TROUT255 format routine.
  - Register 4 - points to the TIDTAB (Trace Id Table) element that you created.
  - Register 5 - points to an open area in an output area. The format routine takes the data contained in the TTE, makes the data printable, and places the resulting printable data into this output area, starting at the location pointed to by R5. The field TLGBSAVE in the \$TLGWORK area (the variable extension area off of the event trace log PCE) points to the beginning of this output area. The maximum size of this output is defined by an equate in \$HASPEQU named TRCLRECL. Therefore, the maximum area that can be saved in this output area is TRCLRECL-1 (minus one for the carriage control). When the output area is full, a call to a routine named TRCPUT can be made to ‘PUT’ this line and obtain a new output area.
  - Register 14 - contains the return address.
  - Register 15 - contains the format routine entry address.

The TRCPUT service routine is an external routine available to installation format routines to “PUT” a formatted output area and obtain a new output area. You can access the TRCPUT service through a \$CALL TRCPUT call out of the HCT in the PADDR.

On entry to the TRCPUT service routine, you must pass the length of the text in Register 0. You can calculate this by taking the ending address in the output area of the installation data and subtracting the

value in TLGBSAVE. Register 15 must contain the address of the TRCPUT service routine and Register 14 must contain the return address (that is, use standard BALR R14,R15 linkage).

On exit, the TRCPUT service routine is returned in register 5 the address of the new output area. The format routine must return to the caller of the installation format routine. Therefore, the format routine must \$STORE R5 upon return from the TRCPUT service routine.

## Creating a trace table using the \$TRACE macro

---

The following example provides an example and explanation of how to create an installation-defined trace record.

To generate a \$TRACE macro to record register information when its identifier is activated (assuming a \$TIDTAB entry has been defined for the identifier and that the identifier is 255), a trace entry point would appear as follows:

```
STM R0,R15,$REGSAVE
label $TRACE ID=255,LEN=16*4,DATA=$REGSAVE,NAME=$USER
```

The STM instruction stores registers 0-15 in storage at location \$REGSAVE. This location is passed to the \$TRACE macro in the DATA= parameter.

The ID= parameter specifies the event trace identifier (255) associated with this trace point (and previously defined in the \$TIDTAB table).

The LEN= parameter specifies the length of the data to be logged. In this case, 4 bytes for each of the 16 registers.

The DATA= parameter points to the location of the data to be logged. In this case, the register's data was stored by a STM instruction into the \$REGSAVE area.

The NAME= parameter specifies the name associated with this macro call. This name (\$USER) can be extracted from the trace table entry and formatted as part of the output for trace ID 255 (by including a formatting routine for ID 255 in the JES2 event trace log processor—HASPEVTL). When specified, this name is used instead of the label on the \$TRACE macro.

The SUBTASK= parameter (which is only valid in the JES2 environment) specifies whether the \$TRACE is issued from the JES2 main task or a subtask environment. This parameter defaults to the environment for which the assembly is defined (as provided on the ENVIRON= keyword of the \$MODULE macro for this module).

## Block extension reuse table (BERT) tables

---

The BERT tables (\$BERTTABs) can add extensions to existing JES2 checkpointed control blocks such as job queue elements (JQEs), or can create new installation-defined checkpointed control blocks.

### BERT control blocks and macros

The MCT table pair MCTBRTTP points to JES2, installation, and dynamic BERT tables. The \$MCT field MCTBRTTH points to the JES2 BERT table. The \$MCT field MCTBRTTU points to the installation table, if such a table exists.

The \$BERTTAB macro builds both the JES2 and installation tables and table elements. This macro also contains the mapping macro for the BERT table and element.

The \$DOGBERT executable macro is used to locate the data define by the \$BERTTABs in the BERTs and collect that data into a control block. For control block types defined by JES2 (such as JQE and CAT), higher level services (\$DOGJQE and \$DOGCAT) should be used instead of \$DOGBERT to manage this data.

The \$DOGBERT executable macro can also be coded with the ACTION=GETOFFSET operand to obtain the offset and length of the data defined by a particular \$BERTTAB.

## A JES2 BERT table

Figure 18 on page 446 illustrates a JES2 BERT table. The table name is HASPBRTT, the same as that specified in the V-type address in the \$MCT. The table is delimited by TABLE=HASP (to start the table) and TABLE=END (to end the table). The table element shown represents all the information JES2 needs to define an extension to the JQE containing accounting information from the JOB card for the job.

When \$BERTTAB is specified with operands other than TABLE=, the macro generates a table element. In Figure 19 on page 446, the table element is generated for JQE accounting information. All JES2-defined BERTs are defined within this single table.

```
HASPBRTT $BERTTAB TABLE=HASP
          $BERTTAB ...
          $BERTTAB CBTYPE=JQE,NAME=ACCT,CBOFF=JQAACCT-JQE,          X
                    LEN=L'JQAACCT
          $BERTTAB ...
          $BERTTAB TABLE=END
```

Figure 18. The JES2 BERT Table

## An installation BERT table

Figure 19 on page 446 illustrates an installation BERT table.

```
USERBRTT $BERTTAB TABLE=USER
          $BERTTAB CBTYPE=JQE,NAME=UNOTIFY,CBOFF=*,          X
                    LEN=8
          $BERTTAB TABLE=END
```

Figure 19. Example of an Installation BERT Table

## A dynamic BERT table

Figure 20 on page 446 illustrates an alternative method of defining the BERT table from Figure 19 on page 446 through use of a dynamic table.

```
MYBRTTB $BERTTAB TABLE=DYNAMIC.
          $BERTTAB CBTYPE=JQE,NAME=UNOTIFY,CBOFF=*,          X
                    LEN=8
          $BERTTAB TABLE=END
```

Figure 20. Example of a Dynamic BERT Table

To create the installation BERT table coded illustrated in Figure 19 on page 446 or Figure 20 on page 446, you need to code the following on the \$BERTTAB macro:

Operand	Description
CBTYPE=JQE	The control block type with which the data is to be associated. In this case the data is to be associated with a JQE. You can also associate your own data with a Class Attribute Table (CAT), WLM Servic Class Queue head (WSCQ), or your own installation control block type. By convention, the JES2 table will not use CBTYPE= values beginning with the letters U or V. Installation CBTYPE= values should therefore begin with one of these two letters to avoid potential conflict with future JES2 types.
NAME=UNOTIFY	A unique name which identifies the specific date within the control block type. By convention, the JES2 table will not use NAME= values beginning with the letters U or V. Installation NAME= values should therefore begin with one of these two letters to avoid potential conflict with future JES2 types.

Operand	Description
CBOFF=*	The offset within the control block of the data defined by this table. CBOFF=* indicates that the offset is to be determined at run time. The \$DOGBERT ACTION=GETOFFSET macro should be coded (with CBTYPE= and NAME= equal to the specification on the \$BERTTAB) to locate the data in the control block.
LEN=8	The length of the data area defined by the \$BERTTAB.

## Coding the other pieces

In addition to coding the installation BERT table, you need to do the following:

- Code Exit 0. Exit 0 code must:
  - Obtain the \$UCT and place the \$UCT's address in the \$HCT.
  - Initialize the \$UCT.
  - Place the installation work selection table address in the MCTBRTTU field in the \$MCT in module HASPTABS. This is not necessary for dynamic tables. Dynamic tables should be linked to the table pair by placing a LOAD initialization statement in your JES2 initialization stream for the module containing the dynamic BERT table.
- Provide routines (such as installation exits) that fill in or use the BERT data. These routines should access the data as follows:
  - For extensions to the JQE (CBTYPE=JQE):
    - Access the JQA using the \$DOGJQE macro.
    - Determine the offset of the data within the JQA using the \$DOGBERT ACTION=GETOFFSET service. The address of the data can then be computed by adding this offset to the JQA address.
  - For extensions to the CAT (CBTYPE=CAT):
    - Access the CAT using the \$DOGCAT macro.
    - Determine the offset of the data within the CAT using the \$DOGBERT ACTION=GETOFFSET service. The address of the data can then be computed by adding this offset to the CAT address.
  - For installation-defined CBTYPE values:
    - Obtain storage to contain the control block and a PREBERT. The PREBERT must precede the control block in this storage. The length of the control block (without the PREBERT) can be obtained using the \$DOGBERT ACTION=GETLENGTH macro.
    - Access the data using the \$DOGBERT macro. Use ACTION=FETCH to read the BERT data from the checkpoint and ACTION=RETURN to return it to the checkpoint.

## JES2 \$SCAN facility

JES2 provides a service facility for scanning parameter statement input (initialization statement and operator commands) called \$SCAN. It is a general facility that defines a general grammar for the input statements to be processed, allows for definition of the allowed input through tables, and provides for special processing through exit routines called during the scan.

\$SCAN is basically designed to perform most of the scanning required for processing the JES2 initialization statements, with the remaining processing for those statements being done by the exits from \$SCAN, and to allow the use of multiple tables to define the allowed parameter input. \$SCAN can scan various input structures, including those that require recursive calls to \$SCAN itself. At each level of recursion, \$SCAN can use two tables of specifications that define the allowed input at that level.

JES2 has implemented the scanning of its initialization options and its initialization statements using \$SCAN and a series of these **table pairs**. A JES2-defined table has been built as the second table of

each pair, and an installation table can be defined as the first table to add to or modify the specifications in the JES2 table. \$SCAN can be useful, as well, in implementing other types of statements within routines called from the \$EXIT facility, such as installation-defined operator commands or JES2 job control statements.

Six macros are provided to aid your use of the \$SCAN facility. They are \$SCAN, \$SCANB, \$SCANCOM, \$SCAND, \$SCANDIA, and \$SCANTAB. It is important that you understand the interrelationships of these macros before attempting to implement any use of the \$SCAN facility.

## \$SCAN-related control blocks

There are several control blocks related to \$SCAN. First, the facility recognizes a set of “primitive” control blocks specified in the scan table entries. They are the HCT, HCCT, the current PCE, the current DTE, DCTs, the user control table (UCT), or any control block pointed to by a name/token pair. The UCT is not generated or specifically used by JES2, but rather is an optional user control block pointed to by the \$UCT field of the HCT. Installations requiring a central control block for use in exit routines or user modifications needs to generate a UCT, or a their own control table pointed to by a name/token pair, and use it as their central main task control block rather than adding new fields to the HCT.

Additionally, scan table entries can indicate the control block from the previous “level” of scanning or a temporary control block should be used. A subsequent search for the actual required control block through control block chains and subscript indexing can also be indicated by the table entries.

Another important control block for the JES2 uses of \$SCAN is the JES2 master control table (MCT). The MCT is pointed to by field \$MCT in the HCT and it contains the addresses of many JES2 statically-defined tables and related routines. Importantly, the MCT contains the doublewords containing addresses of the scan table pairs. The MCT is assembled into module HASPTABS in load module HASJES20.

The scan table entries themselves form control blocks which are mapped by the \$SCANTAB macro. Also, during a scan, a work area is used by \$SCAN and passed to prescan and postscan exit routines. The scan work areas are allocated through \$GETWORK and mapped by the \$SCANWA macro.

## Implementing \$SCAN tables

The \$SCANTAB macro should be used to generate a scan table. Your installation can define, for example, a table that describes the keywords and input allowed on a JES2 job control statement and use \$SCAN and that table from a JES2 HASPRDR Exit 4 to implement your own JES2 job control statements.

As mentioned, the JES2 initialization options, initialization parameter statements, and some operator commands are now implemented using \$SCAN. (See *z/OS JES2 Commands* for a list of these commands.) Scan tables in the HASPTAB module make up the JES2 half of the table pairs that define those options and parameters. Your installation can define its own scan tables to add to or replace any or all of the JES2 scan table entries.

Each of the pairs of table addresses used in the implementation of the initialization statements is defined in the MCT in HASPTABS. You can include your table by using one of the following techniques:

- Specify \$SCANTAB TABLE=(DYNAMIC,*pair-offset*). When the module containing the tables is loaded (through the LOAD(xxxxxxx) initialization statement), the tables will be associated with the specified table pair. Multiple tables may be listed if your tables are to be associated with multiple statements or commands. Valid table pair names take the form of MCTxxxTP, where xxx is listed in [Table 10 on page 450](#).
- Your installation can locate the MCT while running in a JES2 initialization exit (for example, Exit 0) and store the addresses of its tables in the MCTxxxTU fields of the MCT, or you can point to your installation-defined UCT that contains a pair of table addresses.
- The linkage editor can be used to define these table addresses to JES2 by linkediting them (and possibly the UCT) into HASJES20. This is possible because the MCTxxxTU and \$UCT fields are defined as the weak external symbol names. The installation must name its scan tables with those specific names (listed in [Table 10 on page 450](#)) in order to use this method of providing user scan tables for the initialization statements.

The installation tables can provide entries that define new initialization options or statements, new parameters on JES2-defined statements, or new commands. Since the installation table is searched first, JES2-defined entries can be functionally replaced as well. The following scan tables are involved with the JES2 initialization statements. In each case, the xxx described indicates the MCT labels for the table address pair (MCTxxxTP), the installation table address (MCTxxxTU), and the JES2 table address (MCTxxxTH).

To access \$SCAN table pairs at the highest level of scan use:

**Table Name**  
**To Access**

**OPT**

Initialization Options

**MPS**

Main Parameter Statements (initialization/command statements)

**MG**

Message Generation

There are two table pairs that need to be considered for message generation:

**MCTMGTP**

This table pair in the \$MCT processes message is built in the JES2 main task environment.

**CCTMGTP**

This table pair in the \$HCCT processes message is built in other environments.

When creating messages, you need to be aware of which environments the messages will be built in, and add them to the appropriate table pair (possibly both).

Be aware that although the main initialization parameter statement (MPS) tables are used for JES2 initialization statements and for processing the object of several command verbs, not all command verb processing checks the MPS \$SCAN tables before trying to process a command with non-\$SCAN code. Entries in the installation-defined or JES2-defined MPS table are used for initialization processing if:

- the CALLERS= keyword on the \$SCANTAB macro is not specified **or**
- the CALLERS= keyword on the \$SCANTAB macro includes the correct initialization \$SCAN caller id, as follows:

**\$SCAN Caller ID Equate**  
**Initialization Statement Source**

**\$SCIRPL**

Parmlib or Exit 19

**\$SCIRPLC**

Operator Console

Entries in the installation-defined or JES2-defined MPS table are used for the following command verbs if:

- the object of the verb in the command input matches the NAME= keyword on the \$SCANTAB macro (at least for the minimum length required, that is, MINLEN=) **and**
- the CALLERS= keyword on the \$SCANTAB macro explicitly includes the caller ID for the command verb, as follows:

**\$SCAN Caller ID Equate**  
**Command Verb**

**\$SCDCMDS**

\$D or \$DU (long form)

**\$SCDOCMD**

\$DU (short form)

**\$SCSCMDS**

\$T

**\$SCSTCMD**  
 \$\$  
**\$SCPCMDS**  
 \$P  
**\$SCECMDS**  
 \$E  
**\$SCACMDS**  
 \$ADD  
**\$SCRCMDS**  
 \$DEL  
**\$SCLTCMD**  
 \$DO  
**\$SCECMDA**  
 \$E (MEMBER)  
**\$SCZCMDS**  
 \$Z  
**\$SCHCMDS**  
 \$H  
**\$SCRLCMD**  
 \$A  
**\$SCCCMDS**  
 \$C  
**\$SCTOCMD**  
 \$TO  
**\$SCCOCMD**  
 \$CO  
**\$SCPOCMD**  
 \$PO  
**\$SCOCMDS**  
 \$O  
**\$SCLOCMD**  
 Output short display  
**\$SCLCMDS**  
 \$L

If the command verbs shown in the table above are used against objects for which no match is found in the MPS tables, then non-\$SCAN command support is attempted. Using sub-tables you can also affect the sub-operands of JES2-defined MPS tables entries by using the tables by [Table 10 on page 450](#).

[Table 10 on page 450](#) shows the processing that can be affected by the use of \$SCAN tables.

Table 10. JES2 Reserved Master Control Table Names						
\$SCAN Target	Master Control Table Name	Used for MPS Processing			Parent CB	Notes
		INIT	\$D	\$T		
ACTRMT(nnnn)	ACT		.		RSO	
APPL(applid)	APL	.	.	.	APT	\$ADD
BADTRACK	BAD	.				No commands exist
BUFDEF	BUF	.	.	.		
BUFDEF BELOWBUF=	BFH	.	.	.		
BUFDEF EXTBUF=	BFX	.	.	.		



Table 10. JES2 Reserved Master Control Table Names (continued)

\$SCAN Target	Master Control Table Name	Used for MPS Processing			Parent CB	Notes
		INIT	\$D	\$T		
CKPTDEF	CKT	•	•	•		CKPT recovery dialog
CKPTDEF CKPTn=	KPN	•	•	•	HFAM	CKPT recovery dialog
CKPTDEF NEWCKPTn=	EKN	•	•	•	HFAM	CKPT recovery dialog
CKPTDEF VERSIONS=	VKP	•	•	•		
CKPTDEF VOLATILE=	VLТ	•	•	•		
CKPTLOCK	CKL					\$E only; normally not driven by \$SCAN
CKPTSPACE	SPC	•	•	•		
CKPTSPACE CKPT1	CK1		•			
CKPTSPACE CKPT2	CK2		•			
COMPACT	COM	•	•			
CONDEF	CND	•	•	•		
CONNECT	CON	•	•	•	NAT	\$ADD and \$DEL
DEBUG	DBG	•	•	•		
DESTDEF	DST	•	•	•		
DESTID(destname)	DES	•	•	•	RDT	\$ADD and \$DEL
EDS	EDS	•	•	•	HCCT	EDS configuration command and init statement
ESQ	ESQ		•		ESQ	Display EDS queue inventory
ESTBYTE	EBY	•	•	•	EST	
ESTIME	ETM	•	•	•	EST	
ESTLNCT	ELC	•	•	•	EST	
ESTPAGE	EPG	•	•	•	EST	
ESTPUN	EPN	•	•	•	EST	
EXIT(nnn)	XIT	•	•	•	XRT	\$ADD and \$DEL
FSS(fssname)	FSS	•	•	•	FSSCB	\$ADD
INCLUDE	INC	•				
INIT(nnn)	PIT	•	•	•	PIT	\$S, \$P, and \$Z
INITDEF	PAR	•	•	•		
INTRDR	INR	•	•			
JOBCLASS(n)	CAT	•	•	•	CAT	
JOBCLASS(n) XEQCOUNT=	JCX	•	•	•	CAT	
JOB, JOBQ, STC, TSU	JQE		•	•	JQA	\$D, \$T, \$S, \$P, \$E, \$H, \$A, \$C
JOB, JOBQ, STC, TSU	JQE		•	•	JQA	\$D, \$T, \$S, \$P, \$E, \$H, \$A, \$C
JOB CC=	JCC		•	•	JQA	Display and filtering only
JOB SPOOL=	JSP		•	•	JQA	Display and filtering only

Table 10. JES2 Reserved Master Control Table Names (continued)

\$\$SCAN Target	Master Control Table Name	Used for MPS Processing			Parent CB	Notes
		INIT	\$D	\$T		
JOB, JOBQ, STC, TSU	OTP				JQA	\$DO, \$TO, \$CO, \$PO, \$O
JOB, JOBQ, STC, TSU	LOT				JQA	\$L
JOBDEF	JOB	•	•	•		
JOBPRTY(n)	JPY	•	•	•		
LINE <sub>nnnn</sub>	LNE	•	•	•	DCT	\$DU and \$ADD. \$S, \$P, and \$E are processed by \$\$SCAN and MPS tables but do not use LOG subtables.
LINE <sub>nnnn</sub> .device	LIN	•	•	•		\$D, \$T, \$DU are processed by \$\$SCAN, other commands are not.
LINE <sub>nnnn</sub> .JTn	LJT	•	•	•	DCT	\$D, \$T, \$DU are processed by \$\$SCAN, other commands are not.
LINE <sub>nnnn</sub> .JRn	LJR	•	•	•		\$D, \$T, \$DU are processed by \$\$SCAN, other commands are not.
LINE <sub>nnnn</sub> .STn	LST	•	•	•	DCT	\$D, \$T, \$DU are processed by \$\$SCAN, other commands are not.
LINE <sub>nnnn</sub> .SRn	LSR	•	•	•	DCT	\$D, \$T, \$DU are processed by \$\$SCAN, other commands are not.
LOADMOD(modname)	LOD	•	•	•	LMT	
LOGONn	LOG	•	•	•	DCT	\$DU and \$ADD. \$S, \$P, and \$E are processed by \$\$SCAN and MPS tables but do not use LOG subtables.
MASDEF	MAS	•	•	•		
MEMBER(x)	MEM	•	•	•	QSE	
MEMBER(x) LASTART=	STY		•		QSE	
MODULE	MOD	•	•		MIT	USER/dynamic tables not allowed
NETACCT	NET	•	•			
NJEDEF	NJE	•	•	•		
NETSRV   NSV  • • •  DCT	NSV		•	•	•	DCT
NODE( <sub>nnnn</sub> )	NOD	•	•	•	NIT	\$SN command not processed by \$\$SCAN
NODE( <sub>nnnn</sub> ) AUTH=	NAU	•	•	•	NIT	
NODE( <sub>nnnn</sub> ) PASSWORD=	NDP	•	•	•	NIT	
OFFn.device	OFF	•	•	•		\$DU. \$S OFFn.device, \$P OFFn.device not processed by \$\$SCAN

Table 10. JES2 Reserved Master Control Table Names (continued)

\$SCAN Target	Master Control Table Name	Used for MPS Processing			Parent CB	Notes
		INIT	\$D	\$T		
OFFn.JR	OJR	•	•	•	DCT	\$DU. \$S OFFn.JR, \$P OFFn.JR not processed by \$SCAN
OFFn.JT	OJT	•	•	•	DCT	\$DU. \$S OFFn.JT, \$P OFFn.JT not processed by \$SCAN
OFFn.JT MOD=	OJM	•	•	•	DCT	
OFFn.SR	OSR	•	•	•	DCT	\$DU. \$S OFFn.SR, \$P OFFn.SR not processed by \$SCAN
OFFn.ST	OST	•	•	•	DCT	\$DU. \$S OFFn.ST, \$P OFFn.ST not processed by \$SCAN
OFFn.ST MOD=	OSM	•	•	•	DCT	
OFFLOADn	OFL	•	•	•	DCT	\$DU. \$S, \$P, and \$Z not processed by \$SCAN
OPTSDEF	OPD	•	•	•		
OUTCLASS(n)	SCT	•	•	•	SCAT	
OUTDEF	OUT	•	•	•		
OUTPRTY(n)	OPY	•	•	•		
PATH(x)	PTH		•		NIT	
PATH(x) VIA	VIA		•		NITP	Individual path element
PCEDEF	PCD	•	•			
PCE	PCC		•	•	PTAB	
PCE COUNT=	PCN		•		PTAB	
PCE DETAILS=	PDT		•		PTAB	
PRINTDEF	PTD	•	•	•		
PRINTDEF SETPAGE=	SEP	•	•	•		
PROCLIB	PRL	•	•	•		\$ADD, \$DEL
PROCLIB DD=	PDD	•	•	•	PAD	\$ADD, \$DEL
PRTnnnn	PRT	•	•	•	DCT	\$B, \$C, \$E, \$F, \$I, \$N, \$P, \$S, \$Z are not processed by \$SCAN
PRTnnnn FSSINFO=	PRF	•	•	•	DCT	\$d only
PUNnn	PUN	•	•	•	DCT	\$B, \$C, \$E, \$F, \$I, \$N, \$P, \$S, \$Z are not processed by \$SCAN
PUNCHDEF	PUD	•	•			
RDI	RDI		•	•	DCT	
REDIRECT	RED	•	•	•	CRE	\$ADD
Rnnnn.device	RDV	•	•	•		\$B, \$C, \$E, \$F, \$I, \$N, \$P, \$S, and \$Z are not processed by \$SCAN

Table 10. JES2 Reserved Master Control Table Names (continued)

\$SCAN Target	Master Control Table Name	Used for MPS Processing			Parent CB	Notes
		INIT	\$D	\$T		
Rnnnn.PRm	RPR	•	•	•	DCT	\$B, \$C, \$E, \$F, \$I, \$N, \$P, \$S, and \$Z are not processed by \$SCAN
Rnnnn.PUm	RPU	•	•	•	DCT	\$B, \$C, \$E, \$F, \$I, \$N, \$P, \$S, and \$Z are not processed by \$SCAN
Rnnnn.RDm	RRD	•	•	•		\$B, \$C, \$E, \$F, \$I, \$N, \$P, \$S, and \$Z are not processed by \$SCAN
Rnnnn.CON	RCN		•		DCT	
RDRnn	RDR	•	•	•	DCT	\$C, \$P, \$S, \$Z are not processed by \$SCAN
RDRnn AUTH=	RAU	•	•	•	DCT	Also RDI AUTH= and Rn.RDn AUTH=
RECVOPTS(type)	RCV	•	•	•	RVS	
REQJOBID	RQJ	•	•	•		
RMT(nnnn)	RMT	•	•	•	RAT	\$DU, \$\$, \$P commands are processed by \$SCAN and MPS tables but do not use the RMT subtables
SMFDEF	SMF	•	•	•		
SOCKET	SCK		•	•	•	SCK
SPOOL	SPL		•		DAS	\$D, \$S, \$P, \$Z
SPOOL UNITDATA=	VUN		•		DAS	
SPOOLDEF	SPD	•	•	•		
SPOOLDEF FENCE=	FEN	•	•	•	z	
SPOOLDEF TGSPACE=	TGS	•	•	•		
SSI(nnn)	SSI	•	•	•		
SUBNET(xxxxx)	SUB		•		NSACT	
SUBTDEF	SBD	•	•	•		
TPDEF	TPD	•	•	•		
TPDEF BSCBUF=	BSC	•	•	•		
TPDEF SNABUF=	SNA	•	•	•		
TPDEF SESSIONS=	SES		•			
TRACE(nnn)	TRI	•	•			\$\$, \$P commands are processed by \$SCAN
TRACEDEF	TRC	•	•	•		
TRACEDEF LOG=	TLG	•	•	•		
TRACEDEF STATS=	STA	•				
ZAPJOB ZBJ=	INIT	•				ZAPJOB command and init stmt

## Examples of \$SCAN tables

The three examples in [Figure 21 on page 456](#) show:

- (A) how to add a new simple initialization statement
- (B) how to replace a parameter statement specification on the PRT(*nnnn*) statement
- (C) how to define a new initialization statement to provide an installation-specific function with a single parameter.

All the specifications that can be made in scan table entries are described with the \$SCANTAB macro.

Reviewing the JES2 tables in HASPSTAB that define the initialization statements will illustrate to you, most of the capabilities of the \$SCANTAB specifications and the \$SCAN facility.

You can specify the addresses of these tables to JES2 by including the tables in an assembly module linked into HASJES20 or by having an Exit 0 or Exit 19 routine locate the tables and save their addresses in the MCT.

(A)

```
*****
*
*   USER TABLE FOR MAIN INITIALIZATION STATEMENTS
*
*   'USERCAN' - USER DEFINED STATEMENT THAT SHOULD EFFECT THE
*               SAME ACTION AS JES2 'CANCEL'
*
*****
SPACE 1 MYTAB1  $SCANTAB TABLE=DYNAMIC,MCTMPSTP)
$SCANTAB NAME=USERCAN, FIELD=CIRFLAG1, CB=PCE, CONV=FLAG,      C
              VALUE=(, CIRF1CAN, FF), CALLERS=$SCIRPLC
$SCANTAB TABLE=END
```

(B)

```
*****
*
*   USER TABLE FOR PRINTER INITIALIZATION STATEMENTS
*
*   'FORMS' - USER DEFINED REPLACEMENT KEYWORD FOR THE
*             JES2 FORMS PARAMETER - ALLOWS THE INPUT TO
*             BE ONLY 3-5 CHARACTERS LONG RATHER THAN 1-8.
*
*   'USERCLS' - USER DEFINED STATEMENT THAT SHOULD EFFECT THE
*               SAME ACTION AS JES2 'CLASS', EXCEPT ONLY
*               FOR CLASSES A-Z.
*
*****
SPACE 1 MYPRTTAB $SCANTAB TABLE=(DYNAMIC,MCTPRTTTP)
$SCANTAB NAME=FORMS, CB=PARENT, DSECT=DCTDSECT, MINLEN=1,      C
              FIELD=DCTFORMS, CONV=CHARAN, RANGE=(3,5)
$SCANTAB NAME=USERCLS, CB=PARENT, DSECT=DCTDSECT, CONV=CHARA, C
              FIELD=(DCTCLASS, PITCLLEN), RANGE=(1, PITCLLEN-1)
$SCANTAB TABLE=END
```

(C)

```
*****
*
*   USER TABLE TO DEFINE A NEW INITIALIZATION STATEMENT
*   TO DEFINE THE MAXIMUM NUMBER OF VALID USERS
*
*   'USERDEF' - USER DEFINED STATEMENT TO SPECIFY THE
*               NUMBER OF USERS
*
*               NUM - PARAMETER ON THE 'USERDEF'
*                   STATEMENT TO DEFINE THE MAXIMUM
*                   NUMBER OF DEFINED USERS
*
*****
SPACE 1 UDEFTAB $SCANTAB TABLE=(DYNAMIC,MCTMPSTP)
$SCANTAB NAME=USERDEF, CB=UCT, CONV=SUBSCAN,                  C
              SCANTAB=(UCTSCANT, UCT),
              TABLE=END
USUBSCAN $SCANTAB TABLE=USER
$SCANTAB NAME=NUM, CB=UCT, MINLEN=3, DSECT=UCT, FIELD=$UCTNUM, C
              CONV=NUM, RANGE=(0,9)
$SCANTAB TABLE=END
SPACE 1
UCT
.
.
. USERCB EQU
UCTSCNTP $PAIR FORM=TABLE, TABLE=SCAN,
              HASPENT=NONE, USERENT=USUBSCAN
$UCTNUM DS F
.
.
.
```

Figure 21. Three Examples of \$SCANTAB Tables

The above examples are only examples of defining and modifying initialization statements by use of the \$SCAN facility. The source module must, of course, include all standard JES2 statements, such as: \$MODULE and \$MODEND.





## Appendix B. Table pairs coding example

This coding example implements an installation security processor. It is made up of a JES2 initialization Exit 0 and a user extension module named HASPXJ00 which contains the installation security processor, the installation security subtask, and the installation PCE, DTE, trace, work selection, and \$SCAN tables. The example includes sample mapping macros \$SCYWORK, \$SCDWORK, and \$UCT, and the macro \$USERCB which invokes the mapping macros.

**Note:** This code is provided as an example of installation extensions to JES2. The code is not Type 1 supported code of IBM. It is not APARable.

The examples are inter-related to show how the tables can be used together. This is not required. That is, it is not necessary to code a PCE table (create your own processor) *and* code a DTE table (create your own subtask). In fact, it may make no sense for certain applications to design interrelated tables. This example was contrived to show what can be done, not necessarily what should be done.

There are six pieces required for the example used here.

- HASPXJ00 - Installation extension code and tables that are required to create an installation security processor, security subtask, trace id, work selection criteria on the offload sysout transmitter work selection list, and an additional operand on the offload sysout transmitter.
- \$UCT - contains required fields for table generation
- \$SCDWORK - subtask DTE extension to hold fields specific to a security subtask
- \$SCYWORK - processor PCE extension to hold fields specific to a security processor
- \$USERCBS - control block that actually generates the above macros. This control block is known by \$MODULE and is the way to get \$MODULE to generate installation control blocks.
- HASPXIT0 - Exit 0 module that contains EXIT0. This exit initializes the \$MCT with the addresses of the installation tables located in HASPXJ00.

### \$USERCBS - Generates user control blocks

```
MACRO -- $USERCBS - USER CONTROL BLOCK DSECT
$USERCBS
*****
*
*      $USERCBS - USER CONTROL BLOCK DSECT
*
* FUNCTION:
*
*      THIS DSECT IS KNOWN BY $MODULE AND WILL BE USED TO GET ALL
*      INSTALLATION CONTROL BLOCKS EXPANDED WITHOUT HAVE TO
*      MODIFY THE $MODULE MACRO.
*
* USED BY:
*
*      ALL INSTALLATION MODULES TO GENERATE ALL INSTALLATION
*      DEFINED CONTROL BLOCKS. FOR DETAILS ON THE FOLLOWING
*      DATA, SEE THE INDIVIDUAL CONTROL BLOCK DSECTS.
*
*      CREATED BY:  N/A          FREED BY:  N/A
*
*      SUBPOOL:    N/A          KEY:  N/A
*
*      SIZE:  N/A          COMPONENT ID:  CODE EXAMPLE
*
*      POINTED TO BY:  N/A
*
*      FREQUENCY:  N/A
*
*      RESIDENCY:  N/A
*
*      SERIALIZATION:  N/A
*
*      CHANGE ACTIVITY, GUIDE 65 - CHICAGO, ILL - 7/86
```

```

*
*****
GBLC  &TITLEID;
LCLC  &TITL;
USERCBS DSECT      USER CONTROL BLOCK DSECT
&TITL   SETC  '&TITLEID -- $UCT      - USER CONTROL TABLE'
        TITLE '&TITL'
        $UCT      ,      GEN THE UCT
&TITL   SETC  '&TITLEID -- $SCDWORK - SECURITY SUBTASK WORK DSECT'
        TITLE '&TITL'
        $SCDWORK ,      GEN THE SECURITY SUBTASK WORK DSECT
&TITL   SETC  '&TITLEID -- $SCYWORK-SECURITY PCE WORK DSECT'
        TITLE '&TITL'
        &SCYWORK; ,      GEN THE SECURITY PCE WORK DSECT
MEND

```

## \$SCYWORK - Processor work area

```

MACRO -- $SCYWORK -- USER SECURITY PROCESSOR WORK AREA DSECT
$SCYWORK
*****
*
*          $SCYWORK - USER SECURITY PROCESSOR WORK AREA DSECT
*
* FUNCTION:
*
*          HOLD FIELDS UNIQUE TO THE SECURITY PROCESSOR PCE
*
* USED BY:
*
*          ALL SECURITY PROCESSOR PCE(S)
*
*          CREATED BY:  PCEDYN          FREED BY:  PCEDYN
*
*          SUBPOOL:   1                KEY:   1
*
*          SIZE:  SEE SCYLEN EQUATE      COMPONENT ID:  CODE EXAMPLE
*
*          POINTED TO BY:  UCTSYPCE FIELD OF THE $UCT DATA AREA      @MES
*
*          FREQUENCY:  ONE PER SECURITY PCE
*
*          RESIDENCY:  VIRTUAL - ABOVE
*                   REAL - ANYWHERE
*
*          SERIALIZATION:  JES2 MAIN TASK SERIALIZATION
*
*          CHANGE ACTIVITY:  GUIDE 65 - CHICAGO, ILL - 7/86
*                   1/88 - FIXED COMMENT
*
*****
PCE      DSECT      USER SECURITY PROCESSOR WORK AREA
        ORG  PCEWORK  PCE WORK AREA
        SPACE 1
*****
*
*          FIELDS UNIQUE TO THE SECURITY PCE
*
*****
SCYDTEAD DS  A      ADDR OF THE SECURITY DTE
SCYTQE  DS  XL(TQLENG)  HASP TIME QUEU ELEMENT
*  FIELD GOES HERE
*  FIELD GOES HERE
*  FIELD GOES HERE
SCYLEN  EQU  *-PCEWORK  LENGTH OF SCY
MEND

```

## \$SCDWORK - Subtask work area

```

MACRO -- $SCDWORK -- USER SECURITY SUBTASK WORK AREA DSECT
$SCDWORK
*****
*
*          $SCDWORK - USER SECURITY SUBTASK WORK AREA DSECT
*
* FUNCTION:
*

```

```

*
*      HOLD FIELDS UNIQUE TO THE SECURITY SUBTASK
*
*
* USED BY:
*
*      ALL SECURITY SUBTASKS
*
*      CREATED BY:  DTEDYN              FREED BY:  DTEDYN
*
*      SUBPOOL:   1                    KEY:    1
*
*      SIZE:  SEE SCDLEN EQUATE          COMPONENT ID:  CODE EXAMPLE
*
*      POINTED TO BY:  UCTSYDTE FIELD OF THE $UCT DATA AREA    @MES
*
*      FREQUENCY:  ONE PER SECURITY SUBTASK
*
*      RESIDENCY:  VIRTUAL - BELOW
*                  REAL - BELOW
*
*      SERIALIZATION:  SUBTASKS FOLLOW MVS SERIALIZATION CONCERNS
*
*      CHANGE ACTIVITY:  GUIDE 65 - CHICAGO, ILL - 7/86
*                      1/88 - ADD SCDHCT
*
*****
DTE      DSECT      USER SECURITY SUBTASK WORK AREA
      ORG    DTEWORK      DTE WORK AREA
      SPACE 1
*****
*
*      FIELDS UNIQUE TO THE SECURITY SUBTASK
*
*****
SCDHCT  DS    A(*-*)      ADDRESS OF HCT          @SA
*      FIELD GOES HERE
*      FIELD GOES HERE
SCDLEN  EQU  *-DTEWORK      LENGTH OF SCD
      MEND

```

## \$UCT - User communication table

```

      MACRO -- $UCT -- USER COMMUNICATION TABLE DSECT
      $UCT
*****
*
*      $UCT - USER COMMUNICATION TABLE DSECT
*
* FUNCTION:
*
*      HOLD FIELDS VARIABLES COMMON FOR INSTALLATION CODE
*
* USED BY:
*
*      ALL INSTALLATION PROCESSOR/FUNCTIONS CAN MAKE USE OF
*      THE $UCT
*
*      CREATED BY:  HASPXITO          FREED BY:  JES2 TASK TERMINATION
*
*      SUBPOOL:   0                    KEY:    1
*
*      SIZE:  SEE UCTLEN          COMPONENT ID:  CODE EXAMPLE
*
*      POINTED TO BY:  $UCT FIELD OF THE $HCT DATA AREA
*
*      FREQUENCY:  ONE PER JES2 SYSTEM
*
*      RESIDENCY:  VIRTUAL - ABOVE
*                  REAL - ANYWHERE
*
*      SERIALIZATION:  JES2 MAIN TASK SERIALIZATION
*
*      CHANGE ACTIVITY:  GUIDE 65 - CHICAGO, ILL - 7/86
*
*****
UCT      DSECT      USER COMMUNICATION TABLE DSECT
UCTID    DS    CL4'UCT'      UCT IDENTIFIER
UCTSCDE  DS    A(*-*)      ADDRESS OF INSTALLATION LOAD MODULE
      SPACE 1

```

```

*****
*
*           FIELDS REQUIRED FOR THE PCE TABLES
*
*****
      SPACE 1
UCTMSCTY DS  A(*-*)           ADDR OF ENTRY POINT
UCTSYPCP DS  A(*-*)           SECURITY PROCESSORS
UCTSYNUM DS  H'1',H'0'
UCTSYQUE DS  A(*-*)           ADDR OF ELEMENT TO BE VERIFIED
UPCESCTY EQU  255             ID OF SECURITY PCE
$DRSCTY  EQU  63             DISPATCHER SECURITY RESOURCE
      SPACE 1

*****
*
*           FIELDS REQUIRED FOR THE DTE TABLES
*
*****
      SPACE 1
UCTMDSCY DS  A(*-*)           ADDR OF ENTRY POINT
UCTSYDTE DS  A(*-*)           ADDR OF SECURITY DTE
UDTESCTY EQU  255             ID OF SECURITY DTE
      SPACE 1
*****
*
*                               END OF UCT
*
*****
      SPACE 1
UCTLEN    EQU  *-UCT           LENGTH OF UCT
      MEND

```

## EXIT 0 - Initialization

```

XITO      TITLE 'USER EXIT 0 MODULE -- PROLOG (MODULE COMMENT BLOCK)'
*****
*
*  MODULE NAME = HASPXITO CSECT
*
*  DESCRIPTIVE NAME = HASP EXIT 0 INITIALIZATION MODULE
*
*  STATUS = OS/VS2 - SEE $MODULE EXPANSION BELOW FOR FMID, VERSION
*
*  FUNCTION = THE HASPXITO MODULE INITIALIZES THE INSTALLATION $UCT
*              AND OTHER INSTALLATION DEFINED ADDRESSES AND FIELDS.
*
*  NOTES = SEE BELOW
*
*  DEPENDENCIES = 1) JES2 EXIT EFFECTOR
*                 2) JES2 PROCESSOR AND SUBTASK DISPATCHING
*
*  RESTRICTIONS = THIS CODE IS PROVIDED AS AN EXAMPLE OF
*                  INSTALLATION EXTENSIONS TO JES2. THIS CODE IS
*                  NOT TO BE CONSIDERED TYPE 1 SUPPORTED CODE OF
*                  IBM.
*
*  REGISTER CONVENTIONS = R0-R3  = WORK REGISTER
*                         R4      = ADDRESS OF THE MTE ENTRY
*                         R5      = ADDRESS OF THE MCT
*                         R6-R9   = WORK REGISTER
*                         R10     = ADDRESS OF THE UCT
*                         R11     = ADDRESS OF THE HCT
*                         R12     = LOCAL ADDRESSABILITY
*                         R13     = ADDRESS OF THE HASPINIT PCE
*                         R14-R15 = WORK AND LINKAGE REGISTER
*
*  PATCH LABEL = NONE
*
*  MODULE TYPE = CSECT
*
*  PROCESSOR = OS/VS ASSEMBLER H OR ASSEMBLER XF (370)
*
*  MODULE SIZE = SEE $MODEND MACRO EXPANSION AT END OF ASSEMBLY
*
*  ATTRIBUTES = NOT REUSABLE, NON-REENTRANT, SUPERVISOR STATE,
*              PROTECT KEY OF HASP'S (1) OR 0, RMODE 24,
*              AMODE 24/31
*

```

```

* ENTRY POINT = EXIT0
*
* PURPOSE = SEE FUNCTION
*
* LINKAGE = STANDARD JES2 $SAVE/$RETURN LINKAGE
*
*
* INPUT  RO  = A CODE INDICATING WHERE THE INITIALIZATION OPTIONS
*           WERE SPECIFIED
*        R1  = ADDRESS OF A 2-WORD PARAMETER LIST WITH THE
*           FOLLOWING STRUCTURE:
*           WORD 1 (+0):  ADDR OF INITIALIZATION OPTIONS STRING
*           WORD 2 (+4):  LENGTH OF INITIALIZATION OPTIONS STRING
*        R11 = ADDRESS OF HCT
*        R13 = ADDRESS OF INITIALIZATION PCE
*        R14 = RETURN ADDRESS
*        R15 = ADDRESS OF ENTRY POINT
*
* OUTPUT R15 = RETURN CODE
*         (ALL OTHERS UNCHANGED)
*
* EXIT-NORMAL = RETURN TO CALLER (HASPIRMA)
*
* EXIT-ERROR = RETURN TO CALLER (HASPIRMA) WITH NON-ZERO RETURN CODE
*
* EXTERNAL REFERENCES = SEE BELOW
*
*   ROUTINES = MISCELLANEOUS JES2 SERVICE ROUTINES, AND
*               MISCELLANEOUS STANDARD SUPERVISOR SERVICE ROUTINES
*
*   DATA AREAS = SEE $MODULE MACRO EXPANSION
*
*   CONTROL BLOCKS = SEE $MODULE MACRO EXPANSION
*
* TABLES = SEE $MODULE MACRO DEFINITION (BELOW)
*
* MACROS = JES2 - $ENTRY, $GETMAIN, $MODCHK, $RETURN, $SAVE
*
* MACROS = MVS   - NONE
*
* CHANGE ACTIVITY:  GUIDE 65 - CHICAGO, ILL - 7/86
*                   CODE AT SP1.3.6/2.1.5 LEVEL
*                   1/88 - VARIOUS FIXES FOR T.B.
*
*****
*
*   TITLE 'USER XITO INITIALIZATION -- PROLOG ($HASPGBL)'
*   COPY  $HASPGBL                COPY HASP GLOBALS                @133
*
*   TITLE 'HASP XITO INITIALIZATION -- PROLOG ($MODULE)'           @133
*   HASPXITO $MODULE NOTICE=NONE,
*
*           TITLE='HASP XITO INITIALIZATION',
*           $DTE,                GENERATE HASP DTE DSECT
*           $ERA,                GENERATE HASP ERA DSECT
*           $HCT,                GENERATE HASP HCT DSECT
*           $HASPEQU,           GENERATE HASP EQUATES DSECT
*           $MCT,                GENERATE HASP MCT DSECT
*           $MIT,                GENERATE HASP MIT DSECT
*           $MITETBL,           GENERATE HASP MITETBL DSECT
*           $MODMAP,            GENERATE HASP MODMAP DSECT
*           $PCE,                GENERATE HASP PCE DSECT
*           $TQE,                GENERATE HASP TQE DSECT
*           $USERCBS,           GENERATE HASP USERCB DSECT
*           $XECB                GENERATE HASP XECB DSECT
*
*
*   TITLE 'USER XITO INITIALIZATION -- EXIT0 - OBTAIN AND SET
*   NECESSARY INFORMATION'
*
*****
*
*   EXIT0 - INSTALLATION EXIT 0 ROUTINE
*
* FUNCTION:
*
*   THIS EXIT POINT OBTAINS A $UCT CONTROL BLOCK, INITIALIZES
*   IT AND PLACES ITS ADDRESS IN THE $HCT.  THIS ROUTINE ALSO
*   INITIALIZES THE $MCT WITH THE SPECIFIED INSTALLATION TABLE
*   ADDRESSES.

```

```

* LINKAGE:
*
*          CALL BY JES2 INITIALIZATION
*
* ENVIRONMENT:
*
*          JES2 MAIN TASK LIMITED (INITIALIZATION).
*
* RECOVERY:
*
*          NONE
*
* REGISTER USAGE (ENTRY/EXIT):
*
*   REG          VALUE ON ENTRY          VALUE ON EXIT
*
*   R0           WHERE INIT OPTIONS
*                SPECIFIED              UNCHANGED
*   R1           ADDR OF PARM LIST        UNCHANGED
*   R2-R10       N/A                     UNCHANGED
*   R11          HCT BASE ADDRESS         UNCHANGED
*   R12          N/A                     UNCHANGED
*   R13          INIT PCE BASE ADDRESS    UNCHANGED
*   R14          RETURN ADDRESS           UNCHANGED
*   R15          ENTRY ADDRESS            RETURN CODE (SEE BELOW)
*
* PARAMETER LIST:
*
*          +0 - ADDR OF INIT OPTIONS STRING
*          +4 - LENGTH OF INIT OPTIONS STRING
*
* REGISTER USAGE (INTERNAL):
*
*   REG          VALUE
*
*   R0-R3        WORK REGISTERS
*   R4           MTE ENTRY ADDRESS
*   R5           MCT BASE ADDRESS
*   R6-9         WORK REGISTER
*
*   R10          UCT BASE ADDRESS
*   R11          HCT BASE ADDRESS
*   R12          LOCAL BASE ADDRESS
*   R13          INIT PCE BASE ADDRESS
*   R14          LINK/WORK REGISTER
*   R15          LINK/WORK REGISTER
*
* RETURN CODES (R15 ON EXIT):
*
*   0 - PROCESSING SUCCESSFUL (NO ERRORS)
*   12 - PROCESSING FAILED, TERMINATE JES2
*
* OTHER CONSIDERATIONS:
*
*          N/A
*
*****
*   SPACE 1
*   USING UCT,R10          ESTABLISH UCT ADDRESSABILITY
*   SPACE 1
*   $ENTRY BASE=R12        DEFINE HASPXITO ENTRY POINT
*   SPACE 2
*   $SAVE TRACE=NO,NAME=EXIT0  GET NEW SAVE AREA, SAVE REGS
*   LR   R12,R15           ESTABLISH BASE REGISTER
*   CLC  $UCT,$ZEROS       ALREADY OBTAINED $UCT...
*   BNE  XITRET0           YES, RETURN TO JES2
*   EJECT
*
*****
*
*          OBTAIN AND INITIALIZE THE UCT
*
*****
*   SPACE 1
*   $GETMAIN RC,LV=UCTLEN,SP=0,LOC=ANY          OBTAIN THE $UCT
*   LTR  R15,R15          GETMAIN SUCCESSFUL...
*   BNZ  XITGTERR         NO, INDICATE ERROR ALLOCATING STOR
*   SPACE 1
*   LR   R2,R1           SET TO
*   LA   R3,UCTLEN       CLEAR THE
*   SLR  R15,R15         STORAGE FOR
*   MVCL R2,R14          THE $UCT

```

```

SPACE 1
ST R1,$UCT          SET UCT ADDRESS IN $HCT
LR R10,R1           SET UCT ADDRESSABILITY
MVC UCTID,=CL4'UCT'  SET UCT ID
MVC UCTSYNUM,$H1     SET NUMBER OF PCE(S) TO DEFINE
EJECT
*****
*
*      LOAD MODULE THAT CONTAINS THE SECURITY PCE, SECURITY DTE,
*      AND THE NECESSARY TABLES TO INSTALL INSTALLATION TAILORING
*
*****
SPACE 1
L R1,$HASPMP          GET THE HASP MODMAP ADDRESS
ICM R1,B'1111',MAPADDR+MAPJXMOD-MAP(R1)  IF HASPXJ00 IN
BNZ XITMODAD          HASJES20, SKIP LOAD

SPACE 1
$MODCHK NAME='HASPXJ00',LOAD=YES,TEST=(MIT,VERSION),          C
MESSAGE=YES,ERRET=XITGTERR  LOAD THE INSTALLATION MODULE

SPACE 1
LR R1,R0              GET EP ADDRESS IN R1
XITMODAD ST R1,UCTSCDE  SAVE THE LOAD MODULE ADDRESS @MES
EJECT
*****
*
*      SEARCH THROUGH MODULE TO FIND ENTRY POINTS FOR THE SECURITY
*      PCE, SECURITY DTE, PCE TABLE, DTE TABLE, TID TABLE, WORK
*      SELECTION TABLE, AND THE $SCAN TABLE.
*
*****
SPACE 1
USING MTE,R4          ESTABLISH MTE ADDRESSABILITY
USING MCT,R5          ESTABLISH MCT ADDRESSABILITY
SPACE 1
L R5,$MCT             OBTAIN THE MCT ADDRESS
XITOLP L R4,MITENTAD-MIT(,R1)  OBTAIN THE MITABLE ADDRESS
LA R6,XITOTBL1        OBTAIN THE TBL OF ENTRY POINTS ADDR
LA R7,XITOTBLL        GET THE NUMBER OF ENTRIES IN TABLE
CLI MTENAME,X'FF'     FOUND END OF TABLE...
BE XITENDT            YES, GO VERIFY ADDRESSES
XITOMTL LH R1,TBLFLDOF(,R6)  OBTAIN THE OFFSET TO THE FIELD
CLC MTENAME,TBLNAME(R6)  ENTRY IN MIT MATCH REQUEST IN TABLE
BNE XITOTB            NO, INCREMENT TO NEXT TABLE ENTRY
CLC TBLFLDCB(L'TBLFLDCB,R6),$ZEROS  YES,CB THE UCT...
BE XITOUCT            YES, GO SET FIELD ADDRESS IN UCT
ALR R1,R5             SET THE FIELD ADDRESS IN THE MCT
B XITOMVC             GO SET ENTRY ADDRESS IN MCT
SPACE 1
XITOUCT ALR R1,R10        SET FIELD ADDRESS IN THE UCT
XITOMVC MVC 0(4,R1),MTEADDR  MOVE ENTRY ADDR INTO CONTROL BLOCK
B XITOPLC             GO CHECK NEXT MIT ENTRY
SPACE 1
XITOTB LA R6,TBLENTRYL(,R6)  INCREMENT TO NEXT TABLE ENTRY
BCT R7,XITOMTL        CHECK NXT TBL ENTRY AGAINST MITABL
XITOLPC LA R4,MTELEN(,R4)  INCREMENT TO NEXT MITABLE ENTRY
B XITOLP              CONTINUE SEARCH FOR ENTRY POINTS
EJECT
*****
*
*      VERIFY THAT THE NECESSARY ADDRESS HAVE BEEN FOUND
*
*****
SPACE 1
XITENDT LA R6,XITOTBL1      SET THE ADDRESS TO TABLE
LA R7,XITOTBLL        SET THE NUMBER OF ENTRIES

XITCLCLP LH R1,TBLFLDOF(,R6)  OBTAIN THE OFFSET INTO THE CB
CLC TBLFLDCB(L'TBLFLDCB,R6),$ZEROS  CONTROL BLOCK THE UCT...
BE XITUCT            YES, GO CHECK IT
AL R1,$MCT           NO, GET THE MCT FIELD ADDRESS
B XITCLC             GO CHECK IF ADDRESS SET
SPACE 1
XITUCT ALR R1,R10        GET THE UCT FIELD ADDRESS
XITCLC CLC 0(4,R1),$ZEROS  FIELD SET...
BE XITGTERR          NO, EXIT WITH AN ERROR
LA R6,TBLENTRYL(,R6)  BUMP TO NEXT TABLE ENTRY
BCT R7,XITCLCLP      GO CHECK NEXT TABLE ENTRY
SPACE 1
*****

```

```

*
*          SET GOOD RETURN CODE AND RETURN
*
*****
      SPACE 1
XITRET0   SLR   R15,R15          INDICATE GOOD RETURN
          B     XITRET          GO RETURN TO JES2
      SPACE 1
*****
*
*          SET ERROR RETURN AND RETURN TO JES2
*
*****
      SPACE 1
XITGTERR  LA     R15,12          INDICATE ERROR RETURN
      SPACE 1
XITRET    $RETURN TRACE=NO,RC=(R15)  END OF EXIT0 INITIALIZATION
          EJECT
*****
*
*          BUILD THE TABLE OF ENTRY POINTS THAT ARE TO BE FOUND.
*          THE TABLE CONSISTS OF:
*
*          CL8'NAME OF ENTRY POINT',
*          AL2(OFFSET INTO EITHER UCT OR MCT OF FIELD TO SET)
*          AL2(0 IF UCT OR 1 IF MCT)
*
*****
      SPACE 1
      DS     0F
XIT0TBL1  DC     CL8'USCTPCE',AL2(UCTMSCTY-UCT),AL2(0)
          DC     CL8'USCTDTE',AL2(UCTMDSCY-UCT),AL2(0)
          DC     CL8'USERPCET',AL2(MCTPCETU-MCT),AL2(1)
          DC     CL8'USERDTET',AL2(MCTDTETU-MCT),AL2(1)
          DC     CL8'USERTIDT',AL2(MCTTIDTU-MCT),AL2(1)
          DC     CL8'USERSTWT',AL2(MCTSTWTU-MCT),AL2(1)
          DC     CL8'USEROSTT',AL2(MCTOSTTU-MCT),AL2(1)
XIT0TBLL  EQU    (*-XIT0TBL1)/12      CALC NUMBER OF ENTRIES
      SPACE 1
TBLNAME   EQU    0,8                NAME OF ENTRY POINT
TBLFLDOF  EQU    8,2                FIELD OFFSET
TBLFLDCB  EQU    10,2               FIELD CONTROL BLOCK
TBLENTYL  EQU    12                LENGTH OF TABLE ENTRY
*****
          TITLE 'HASP XIT0 INITIALIZATION -- EPILOG ($MODEND)'
          $MODEND ,
APARNUM    DC     CL7'XXXXXX'        APAR NUMBER
          END      ,                END OF HASPXITO

```

## User extension code and tables

```

XJ00      TITLE 'USER EXTENSION MODULE -- PROLOG (MODULE COMMENT BLOCK)'
*****
*
*  MODULE NAME = HASJES20 ( HASPXJ00 CSECT )
*
*  DESCRIPTIVE NAME = HASPXJ00 CSECT OF JES2 MAIN MODULE
*
*  STATUS = OS/VS2 - SEE $MODULE EXPANSION BELOW FOR FMID, VERSION
*
*  FUNCTION = THE HASPXJ00 CSECT CONTAINS THE INSTALLATION SECURITY
*              PROCESSOR, THE INSTALLATION SECURITY SUBTASK, AND
*              THE INSTALLATION PCE, DTE, TRACE, WORK SELECTION,
*              AND $SCAN TABLES.
*
*  NOTES = SEE BELOW
*
*  DEPENDENCIES = JES2 PROCESSOR AND SUBTASK DISPATCHING
*
*  RESTRICTIONS = THIS CODE IS PROVIDED AS AN EXAMPLE OF
*                  INSTALLATION EXTENSIONS TO JES2. THIS CODE IS
*                  NOT TO BE CONSIDERED TYPE 1 SUPPORTED CODE OF
*                  IBM.
*
*  REGISTER CONVENTIONS = SEE ENTRY POINT DOCUMENTATION
*
*  MODULE TYPE = PROCEDURE, TABLE ( CSECT TYPE )
*

```



```

*   PROCESSOR = OS/V5 ASSEMBLER H OR ASSEMBLER XF (370)
*
*   MODULE SIZE = SEE $MODEND MACRO EXPANSION AT END OF ASSEMBLY
*
*   ATTRIBUTES = HASP REENTRANT, RMODE 24, AMODE 24/31.
*
* ENTRY POINT =  USCTPCE - INITIAL ENTRY TO SECURITY PROCESSOR
*                 USCTDTE - INITIAL ENTRY TO THE SUBTASK USED FOR
*                 AUTHORIZATION CHECKES
*                 USERPCET - ENTRY FOR INSTALLATION PCE TABLE
*                 USERDTET - ENTRY FOR INSTALLATION DTE TABLE
*                 USERTIDT - ENTRY FOR INSTALLATION TRACE ID TABLE
*                 USERSTWT - ENTRY FOR INSNTALLATION OFFLOAD SYSOUT
*                 TRANSMITTER WORK SELECTION TABLE
*                 USEROSTT - ENTRY FOR INSTALLATION OFFLOAD SYSOUT
*                 TRANSMITTER OPERAND TABLE

```

```

*   PURPOSE = SEE FUNCTION
*
*   LINKAGE = SEE ENTRY POINT DOCUMENTATION
*
*   INPUT = SEE ENTRY POINT DOCUMENTATION
*
*   OUTPUT = SEE ENTRY POINT DOCUMENTATION
*
*   EXIT-NORMAL = SEE ENTRY POINT DOCUMENTATION
*
*   EXIT-ERROR = SEE ENTRY POINT DOCUMENTATION
*
* EXTERNAL REFERENCES = SEE BELOW
*
*   ROUTINES = NONE
*
*   DATA AREAS = SEE $MODULE MACRO SPECIFICATION
*
*   CONTROL BLOCKS = SEE $MODULE SPECIFICATION
*
*   TABLES = SEE $MODULE MACRO SPECIFICATION
*
* MACROS = JES2 - $ACTIVE, $AMODE, $CALL, $DECODE, $DORMANT, $DTEDYN,
*                $ENTRY, $MODULE, $PCETAB, $REGS, $RETURN, $SAVE,
*                $SCANTAB, $STIMER, $STORE, $TIDTAB, $TRACE, $WAIT,
*                $WSTAB
*
* MACROS = MVS - ATTACH, DEQ, ENQ, ESTAE, POST, SDUMP, WAIT
*
* CHANGE ACTIVITY:  GUIDE 65, CHICAGO, ILL - 7/86
*                  CODE AT SP1.3.6/2.1.5 LEVEL
*                  1/88 VARIOUS FIXES BY BDB, SA, JK, MES, SWW FOR TB
*
*****

```

```

                TITLE 'USER EXTENSION MODULE -- PROLOG ($HASPGBL)'
                COPY $HASPGBL COPY HASP GLOBALS
                TITLE 'USER EXTENSION MODULE -- PROLOG ($MODULE)'
HASPJX00 $MODULE NOTICE=NONE,
        ENTRIES=(USERPCET,USERDTET,USERTIDT,USERSTWT,USEROSTT),
        TITLE='USER EXTENSION MODULE',
        $DCT, GENERATE HASP DCT DSECT
        $DTE, GENERATE HASP DTE DSECT
        $DTETAB, GENERATE HASP DTETAB DSECT
        $ERA, GENERATE HASP ERA DSECT
        $HASPEQU, GENERATE HASP EQUATES DSECT
        $HCT, GENERATE HASP HCT DSECT
        $JQE, GENERATE HASP JQE DSECT
        $MIT, GENERATE HASP MIT DSECT
        $PCE, GENERATE HASP PCE DSECT
        $PCETAB, GENERATE HASP PCETAB DSECT
        $RDRWORK, GENERATE HASP RDRWORK DSECT

```

```

        $SCANTAB, GENERATE HASP SCANTAB DSECT
        $SCAT, GENERATE HASP SCAT DSECT
        $TIDTAB, GENERATE HASP TIDTAB DSECT
        $TLGWORK, GENERATE HASP TLGWORK DSECT
        $TQE, GENERATE HASP TQE DSECT
        $TRP, GENERATE HASP TRP DSECT
        $TTE, GENERATE HASP TTE DSECT
        $USERCBS, GENERATE USER DSECTS
        $WSTAB, GENERATE HASP WSTAB DSECT

```

```

$XECB                GENERATE HASP XECB DSECT
TITLE 'USER EXTENSION MODULE -- INTRO - BRIEF OVERVIEW OF
      FUNCTION AND RELATED PIECES'
*****
*
* FUNCTION -- THIS MODULE CONTAINS THE INSTALLATION EXTENSION CODE
*              AND TABLES THAT ARE REQUIRED TO CREATE AN INSTALLATION
*              SECURITY PROCESSOR, SECURITY SUBTASK, TRACE ID, WORK
*              SELECTION CRITERIA ON THE OFFLOAD SYSOUT TRANSMITTER
*              WORK SELECTION LIST, AND AN ADDITIONAL OPERAND ON THE
*              OFFLOAD SYSOUT TRANSMITTER.
*
* REQUIRED PIECES -- HASPXJ00 - THIS MODULE
*                   $UCT      - CONTAINS REQUIRED FIELDS FOR TABLE
*                             GENERATION
*                   $SCDWORK - SUBTASK DTE EXTENSION TO HOLD FIELDS
*                             SPECIFIC TO A SECURITY SUBTASK
*                   $SCYWORK - PROCESSOR PCE EXTENSION TO HOLD
*                             FIELDS SPECIFIC TO A SECURITY
*                             PROCESSOR
*                   $USERCBS - CONTROL BLOCK THAT ACTUALLY GENERATES
*                             THE ABOVE MACROS. THIS CONTROL BLOCK
*                             IS KNOWN BY $MODULE AND IS THE WAY
*                             FOR AN INSTALLATION TO GET $MODULE TO
*                             GENERATE THEIR CONTROL BLOCKS
*                   HASPXITO - EXIT 0 MODULE THAT CONTAINS EXIT0.
*                             THIS EXIT INITIALIZES THE $MCT WITH
*                             THE ADDRESSES OF THE INSTALLATION
*                             TABLES LOCATED IN HASPXJ00.
*
*****

```

## USCTPCE - INITIAL ENTRY POINT

```

TITLE 'USER EXTENSION MODULE -- USCTPCE - SECURITY PROCESSOR, C
      INITIAL ENTRY POINT'
*****
*
* PROCESSOR NAME -- USCTPCE
*
* DESCRIPTIVE NAME -- USER SECURITY PROCESSOR
*
* FUNCTION -- MANAGE THE INSTALLATION SECURITY SAF CALLS BY PASSING
*              A REQUEST TO THE SECURITY PROCESSOR'S SECURITY
*              SUBTASK TO ISSUE THE SAF CALL.
*
* NOTES --     BECAUSE A JES2 PROCESSOR IS NOT ALLOWED TO DIRECTLY
*              ISSUE AN OS WAIT, USCTPCE ATTACHES A SUB-TASK TO
*              PERFORM THOSE FUNCTIONS REQUIRING WAITS, THE SUB-TASK,
*              USCDTE, PERFORMS THE CALL TO THE SECURITY
*              AUTHORIZATION FACILITY (SAF).
*
*
* REGISTER CONVENTIONS -- R0 - R2 -- WORK REGISTERS
*                          R3      -- ADDRESS OF $DTE
*                          R4      -- ADDRESS OF WORK ELEMENT
*                          R5 - R9 -- WORK REGISTERS
*                          R10     -- ADDRESS OF $UCT
*                          R11     -- ADDRESS OF $HCT
*                          R12     -- BASE ADDRESSABILITY
*                          R13     -- ADDRESS OF PCE
*                          R14     -- LINKAGE REGISTER
*                          R15     -- LINKAGE REGISTER
*
*****
EJECT
*****
*
*              USCTPCE INITIAL ENTRY POINT
*
*****
SPACE 2
USING UCT,R10          ESTABLISH UCT ADDRESSABILITY
SPACE 1
USCTPCE $ENTRY BASE=R12  PROVIDE PROCESSOR ENTRY POINT
SPACE 1
L      R10,$UCT        OBTAIN THE UCT ADDRESS
EJECT

```

```

*****
*
*          MAIN LOOP OF THE SECURITY PROCESSOR
*
*****
SPACE 1
USCTYLOP $ACTIVE          INDICATE PROCESSOR ACTIVE
1
    BZ    USCATACH        NO, GO ATTACH IT
    TM    DTEFLAG1-DTE(R3),DTE1ACTV  SUBTASK ACTIVE...
    BO    USCTEST         YES, GO QUEUE UP MEMBER
    SPACE 1
*****
*
*          DETACH THE SECURITY SUBTASK (ABENDED)
*
*****
SPACE 1
$DTEDYN DETACH,ID=UDTESCTY,DTE=(R3),WAIT=XECB          C
    XC    SCYDTEAD,SCYDTEAD  DETACH ABENDED SUB-TASK
    EJECT          CLEAR DTE ADDR
*****
*
*          (RE)-ATTACH THE SECURITY SUBTASK
*
*****
USCATACH $DTEDYN ATTACH,ID=UDTESCTY,WAIT=XECB,ERRET=USCATERR          C
    ST    R1,SCYDTEAD        STORE SUBTASK DTE ADDRESS
    MVC    XECBECB-XECB+DTEIXECB-DTE(,R1),$ZEROS CLEAR          C
    LR    R3,R1              COMMUNICATION ECB
    ST    R11,SCDHCT(,R3)    SET THE SUBTASK DTE ADDRESS
    ST    R11,SCDHCT(,R3)    STORE HCT ADDRESS IN DTE XTNSN @SA
*****
*
*          DETERMINE IF THERE IS WORK TO BE DONE
*
*****
SPACE 1
USCTEST  ICM  R4,B'1111',UCTSYQUE  ANYWORK TO DO...
    BNZ    USCWORK                YES, GO DO IT
    SPACE 1
    $DORMANT                      INDICATE THAT PROCESSOR COMPLETE
    SPACE 1
    $WAIT SCTY,INHIBIT=NO        WAIT FOR WORK
    B      USCTYLOP              GO CHECK FOR WORK TO DO
    EJECT
*****
*
*          SET UP FOR SUB-TASK TO PROCESS JOB
*
*
*  INSTALLATION CODE WOULD GO HERE TO PASS TO SUBTASK THE NECESSARY
*  INFORMATION (THROUGH THE DTE EXTENSION THAT IS UNIQUE FOR THE
*  SECURITY SUBTASK).
*
*****
SPACE 1
USCWORK  DS    0H
    XC    UCTSYQUE                INDICATE WORK BEING PROCESSED (IN          C
    REALITY THIS WOULD PROBABLY UNCHAIN          C
    THE REQUEST, NOT CLEAR THE QUEUE)
    EJECT
*****
*
*  MVS POST THE SUBTASK FOR WORK TO DO AND $WAIT FOR IT TO
*  COMPLETE. NOTE THAT THE CALL TO THE SUBTASK IS $TRACE'D,
*  IF TRACING IS ACTIVE.
*
*****
SPACE 1
MVC    XECBECB-XECB+DTEIXECB-DTE(,R3),$ZEROS CLEAR ECB          C
    LA    R1,DTEWECB-DTE(,R3)    POINT TO THE WORK ECB
    SPACE 1
    POST  (1)                    POST SECURITY SUBTASK FOR WORK
    SPACE 1
    $TRACE ID=255,LEN=USCSAFML,OFF=USCTROFF,NAME=SAFCALL

```

```

MVC 0(USCSAFML,R1),USCSAFM SET INFORMATION TO BE TRACED
SPACE 1
USCTROFF LR R1,R3 GET DTE ADDRESS
$WAIT OPER,XECB=DTEIXECB-DTE(,R1) $WAIT FOR SUB-TASK C
TO POST US
EJECT
*****
*
* SUBTASK HAS POSTED US BACK
*
* INSTALLATION CODE WOULD GO HERE TO VALIDATE THE SUCCESS OF THE
* SECURITY CALL AND TO DO ANY PROCESSING RELEVANT TO THE SUCCESS
* OR FAILURE OF THE CALL.
*
*****
SPACE 1
DS 0H VALIDATE THE RESULT OF THE SECURITY C
CALL.
SPACE
*****
*
* BRANCH TO OBTAIN THE NEXT ITEM TO VERIFY
*
*****
SPACE 1
B USCTEST GO CHECK FOR MORE WORK
EJECT

*****
*
* AN ERROR WAS ENCOUNTERED ON THE ATTACH OF THE SUBTASK.
* WAIT FOR 30 SECONDS AND ATTEMPT TO TRY AGAIN.
*
*****
SPACE 1
USCATERR LA R1,SCYTQE GET ADDRESS OF PCE TQE
LA R0,30 SET TIME INTERVAL
ST R0,TQETIME(,R1) IN TQE
ST R13,TQEPCE(,R1) STORE PCE ADDRESS IN TQE
$TIMER (41) CHAIN THIS TQE
$WAIT WORK AND WAIT FOR INTERVAL TO ELAPSE
B USCATCH GO ATTACH SUBTASK
SPACE 1
*****
*
* LIST LITERALS AND SUSPEND ADDRESSABILITIES.
*
*****
SPACE 1
LTORG
SPACE 1
DROP R10,R12,R13 SUSPEND UCT, BASE, AND PCE ADDRESS

```

## USCTDTE - SECURITY SUBTASK, INITIAL ENTRY POINT

```

TITLE 'USER EXTENSION MODULE -- USCTDTE - SECURITY SUBTASK, C
INITIAL ENTRY POINT'
*****
*
* USCTDTE - USER SECURITY SUBTASK
*
* FUNCTION:
*
* THIS IS AN EXAMPLE OF A USER CODED SECURITY SUBTASK. THIS
* SUBTASK IS DEFINED BY THE USERDTE DTE TABLE. THIS SUBTASK
* IS ATTACHED BY THE USCTPCE SECURITY PROCESSOR. THE
* PURPOSE OF THIS SUBTASK IS TO CODE THE SAF CALL TO VERIFY
* THE ELEMENT THAT WAS PASSED TO IT FROM THE SECURITY
* PROCESSOR.
*
* LINKAGE:
*
* CONTROL GIVEN BY MVS VIA AN ATTACH MVS CALL.
*
* ENVIRONMENT:
*
* JES2 SUBTASK
*

```

```

* RECOVERY:
*
*           MVS ESTAE ESTABLISH UPON ENTRY.  THE RECOVERY ROUTINE IS
*           PROVIDED BY THE $STABEND ROUTINE LOCATED IN HASPRAS.
*
* REGISTER USAGE (ENTRY/EXIT):
*
*   REG          VALUE ON ENTRY          VALUE ON EXIT
*
*   R0           N/A                     UNPREDICTABLE
*   R1           DTE ADDRESS AS SPECIFIED ON THE ATTACH CALL  UNPREDICTABLE
*
*   R2-R14       N/A                     UNPREDICTABLE
*   R15          ENTRY ADDRESS           UNPREDICTABLE
*
* PARAMETER LIST:
*
*           ALL NECESSARY INFORMATION LOCATED IN THE DTE, AS PASSED
*           BY THE ATTACHING PROCESSOR.
*
* REGISTER USAGE (INTERNAL):
*
*   REG          VALUE
*
*   R0-R10       WORK REGISTERS
*   R11          HCT BASE ADDRESS
*   R12          LOCAL BASE ADDRESS
*   R13          DTE BASE ADDRESS
*   R14          LINK/WORK REGISTER
*   R15          LINK/WORK REGISTER

```

```

*
* RETURN CODES (R15 ON EXIT):
*
*           N/A
*
* OTHER CONSIDERATIONS:
*
*           N/A
*
*****
SPACE 1
USCTDTE  USING HCT,R11          ESTABLISH HCT ADDRESSABILITY
          USING DTE,R13          ESTABLISH DTE ADDRESSABILITY
          SPACE 1
          $ENTRY BASE=R12        USER SECURITY SUB-TASK
          LR   R12,R15           SET LOCAL BASE
          LR   R13,R1            SET DTE BASE
          L    R11,SCDHCT        SET HCT BASE @SA
***** @BDB
*USCXA   $AMODE 31,RELATED=(USC37)  FORCE 31-BIT MODE FOR UDTE SCTY @BDB
*
* REMOVED THE $AMODE BECAUSE THE $MODULE ENVIRONMENT IS JES2. @BDB
* THIS CAUSES THE EXPANSION TO GENERATE A CONSTANT $HIBITON @BDB
* WHICH RESIDES IN THE HCT.  SINCE WE DON'T AUTOMATICALLY @BDB
* HAVE ADDRESSABILITY TO THE HCT IN A SUBTASK WE ABEND IN @BDB
* EXECUTION. @BDB
* THIS IS NOT A PROBLEM IF THIS ROUTINE IS COPIED INTO ITS @BDB
* OWN MODULE AND THEN CODE THE $MODULE WITH ENVIRON=SUBTASK. @BDB
* @BDB
*****
USCXA   LA    R15,USCXA01        PSEUDO $AMODE $AMODE @BDB
          0    R15,HIGHON        SET HI BIT ON $AMODE @BDB
          BSM  R0,R15            SET MODE $AMODE @BDB
HIGHON  DC    0F'0',X'80000000'  MASK FOR 31 BIT MODE $AMODE @BDB
USCXA01 DS    0H                RESUME $AMODE @BDB
          SPACE 1
*****
*
* SET THE RETRY ROUTINE, THE CLEAN-UP ROUTINE, AND THE
* VRA EXIT ROUTINE ADDRESSES.
*
* INSTALLATION SHOULD SET THE DTERTXAD, DTEESXAD, AND DTEVRXAD
* FOR THE RETRY ROUTINE ADDRESS, THE CLEAN-UP ROUTINE ADDRESS
* AND THE VRA EXIT ROUTINE ADDRESS RESPECTIVELY, IF THESE
* ROUTINES ARE NEEDED.
*
*****
SPACE 1
L       R2,$STABNDA             GET SUBTASK ESTAE RTN ADDRESS

```

LR R3,R13 COPY DTE ADDRESS  
EJECT

```
*****
*
*   E S T A B L I S H   E S T A E   E N V I R O N M E N T
*
*****
SPACE 1
MVC DTEAWRKA(USCSTLN),USCABND MOVE ESTAE PARM LIST
SPACE 1
ESTAE (2),PARAM=(3),RECORD=YES,MF=(E,DTEAWRKA) C
ESTABLISH RECOVERY ENVIRONMENT
SPACE 1
OI DTEFLAG1,DTE1ACTV SHOW SUBTASK ACTIVE
*
* INSTALLATION SHOULD INITIALIZE THE DTE EXTENSION FOR THE SUBTASK
* HERE
*
```

## USCTDTE - SECURITY SUBTASK, MAIN PROCESSING

```
TITLE 'USER EXTENSION MODE -- SECURITY SUBTASK, C
MAIN PROCESSING'
*****
*
*   NOTIFY PROCESSOR THAT WORK NEEDED AND WAIT FOR A RESPONSE
*
*****
USCPOST SPACE 1
XC DTEWECB,DTEWECB CLEAR WORK ECB
SPACE 1
POST DTEIXECB POST PROCESSOR FOR WORK
SPACE 1
TM DTEFLAG1,DTE1TERM SUBTASK SHUTDOWN REQUESTED...
BO USCRET YES, EXIT TO DELETE SECURITY SUBT
SPACE 1
WAIT ECB=DTEWECB ELSE WAIT FOR WORK TO DO
SPACE 1
TM DTEFLAG1,DTE1TERM SUBTASK SHUTDOWN REQUESTED...
BO USCRET YES, EXIT TO DELETE SECURITY SUBT
EJECT
*****
* @BDB
* ISSUE A MVS WTO TO INDICATE THAT THE SUBTASK IS @BDB
* EXECUTING. @BDB
* @BDB
*****
SPACE 1
LA R1,USMSG901 @BDB
WTO MF=(E,(1)) @BDB
SPACE 1
*****
*
* GO POST THE PROCESSOR FOR WORK
*
*****
SPACE 1
B USCPOST GO POST PROCESSOR FOR WORK
```

## USCTDTE - SECURITY SUBTASK, TERMINATION

```
TITLE 'USER EXTENSION MODULE -- SECURITY SUBTASK, C
TERMINATION'
*****
*
*   TERMINATE SECURITY SUBTASK
*
* NOTE THAT THE MAIN TASK TERMINATION CODE WAITS 30 SECONDS
* FOR THE SUBTASK TO GO AWAY BEFORE CONTINUING. IF THE MAIN
* TASK COMPLETES TERMINATION BEFORE THE SUBTASK DOES (DUE TO
* DEBUG TRACING IN THE SUBTASK), AND A03 ABEND WILL RESULT.
*
*****
SPACE 1
USCRET DS 0H
```

```

USC37    $AMODE 24,RELATED=(USCXA)    AMODE 24 FOR SECURITY TERMINATION
        SPACE 1
        ESTAE 0                      CANCEL ESTAE
        SVC   3                      THEN RETURN TO SYSTEM
        EJECT
*****
*
*      CREATE THE ESTAE PARAMETER LIST AND TRACED INFORMATION
*
*****
        SPACE 1
USCABND  ESTAE ,CT,PURGE=NONE,ASYNCH=YES,TERM=NO,MF=L
USCSTLN  EQU  *-USCABND              LENGTH OF ESTAE PARAMETER LIST
        SPACE 1
USCSAFM  DC    C'THIS IS TRACE DATA THAT SHOULD BE FILLED IN FOR
        C'INSTALLATION USE IN TRACING SECURITY CALLS'
USCSAFML EQU  *-USCSAFM
        SPACE 1
$MID     901                        @BDB
USMSG901 WTO  '$MID. SECURITY SUBTASK INVOKED',      @BDBC
        MF=L,ROUTCDE=10,DESC=6                @BDB
        SPACE 1
        DROP R13                      DROP DTE ADDRESSABILITY

```

## TROUTE255 - TRACING ROUTINE FOR SAF CALL

```

                TITLE 'USER EXTENSION MODULE -- TROUT255 - TRACING ROUTINE      C
                FOR SAFCALL ID=255'
*****
*
*      TROUT255 - TRACING ROUTINE IN SUPPORT OF THE TRACE ID 255.
*
* FUNCTION:
*
*      THIS ROUTINE WILL BE CALLED TO FORMAT THE TRACE RECORD FOR
*      THE INSTALLATION TRACE ID 255.  THIS ROUTINE SHOULD BE
*      ALTERED BY THE INSTALLATION TO FORMAT THE INFORMATION THAT
*      WAS SAVED ON THE TRACING OF THIS ID.
*
* LINKAGE:
*
*      BALR R14,15 TO BY HASPMISC
*
* ENVIRONMENT:
*
*      THIS ENVIRONMENT IS CALL FROM THE JES2 MAIN TASK
*
* RECOVERY:
*
*      NONE.
*
* REGISTER USAGE (ENTRY/EXIT):
*
*      REG          VALUE ON ENTRY          VALUE ON EXIT
*
*      R0           N/A                     UNCHANGED
*      R1           TRACE TABLE BUFFER ADDR UNCHANGED
*      R2           TRACE TABLE ENTRY (TTE) UNCHANGED
*      R3           N/A                     UNCHANGED
*      R4           TRACE ID TABLE ENTRY   UNCHANGED
*      R5           POINTER TO REMAINING OUT- POINTER TO LOCATION OUT-
*                  PUT AREA IN PRINT RECORD PUT AREA AFTER THIS ENTRY*
*      R6-R10       N/A                     UNCHANGED
*      R11          HCT BASE ADDRESS         UNCHANGED
*      R12          N/A                     UNCHANGED
*      R13          PCE BASE ADDRESS         UNCHANGED
*      R14          RETURN ADDRESS          UNCHANGED
*      R15          ENTRY ADDRESS           0
*
* PARAMETER LIST:
*
*      NONE.
*
* REGISTER USAGE (INTERNAL):
*
*      REG          VALUE
*
*      R0-R1        WORK REGISTERS

```

*	R2	TTE ADDRESS	*
*	R3	LOCATION IN TTE	*
*	R4	WORK REGISTER	*
*	R5	LOCATION IN OUTPUT AREA	*
*	R6-R8	WORK REGISTER	*
*	R9	*** RESERVED ***	*
*	R10	WORK REGISTER	*
*	R11	HCT BASE ADDRESS	*
*	R12	LOCAL BASE ADDRESS	*
*	R13	PCE BASE ADDRESS	*
*	R14	LINK/WORK REGISTER	*
*	R15	LINK/WORK REGISTER	*
*	RETURN CODES (R15 ON EXIT):		*
*			*
*	0 - PROCESSING SUCCESSFUL (NO ERRORS)		*
*	OTHER CONSIDERATIONS:		*
*			*
*	MUST RETURN THE NEW VALUE OF R5 ON EXIT (I.E., \$STORE (R5))		*
*			*
*****			
	SPACE 1		
	USING TTE,R2	ESTABLISH TTE ADDRESSABILITY	
	USING PCE,R13	ESTABLISH PCE ADDRESSABILITY	
	SPACE 1		
TROUT255	\$ENTRY BASE=R12	ID=255 TRACE FORMATTER ROUTINE	
	\$SAVE NAME=TROUT255,TRACE=NO	SAVE CALLERS REGISTERS	
	LR R12,R15	ESTABLISH BASE ADDRESS	
	SPACE 1		
	LA R3,TTEDATA	POINT TO THE TTE DATA	
	MVC 0(USCSAFML,R5),0(R3)	SET THE TRACED INFO IN OUTPUT AREA	
	LA R0,USCSAFML(,R5)	POINT BEYOND INFORMATION	
	SL R0,TLGBSAVE	AND FIND LENGTH OF PRINT LINE	
	L R15,\$TRCPUT	GET TRCPUT ROUTINE ADDRESS AND@BDB	
	\$CALL (R15)	GO PRINT THE LINE @JK	
	\$STORE (R5)	INSURE NEW BUFFER IS PASSED BACK	
	SPACE 1		
	\$RETURN TRACE=NO	RETURN TO CALLER	
	SPACE 1		
	\$DROP R2,R12,R13	SUSPEND TTE,LOCAL,AND PCE ADDRESS	

## WSTRKGRP - WORK SELECTION ROUTINE

TITLE 'USER EXTENSION MODULE -- WSTRKGRP - WORK SELECTION			C
ROUTINE FOR TRKGRP CRITERIA'			
*****			
*	WSTRKGRP - WORK SELECTION ROUTINE TO COMPARE THE DCT'S		*
*	AND JQE'S NUMBER OF TRACK GROUPS		*
*			*
*	FUNCTION:		*
*			*
*	THIS ROUTINE WILL BE CALLED TO INSURE THAT THE JOB'S NUMBER		*
*	OF TRACK GROUPS IS EQUAL TO OR BEYOND THE DCT'S THRESHOLD.		*
*			*
*	LINKAGE:		*
*			*
*	BALR R14,15 TO BY HASPSERV		*
*			*
*	ENVIRONMENT:		*
*			*
*	THIS ENVIRONMENT IS CALL FROM THE JES2 MAIN TASK.		*
*			*
*	RECOVERY:		*
*			*
*	NONE.		*
*			*
*	REGISTER USAGE (ENTRY/EXIT):		*
*			*
*	REG	VALUE ON ENTRY	VALUE ON EXIT
*			*
*	R0	N/A	UNCHANGED
*	R1	N/A	UNPREDICATABLE
*	R2	ADDR OF CRITERION BEING	
*		PROCESSED	UNCHANGED
*	R4-R5	N/A	UNCHANGED



```

* R6          N/A          UNPREDICTABLE      *
* R7          COMPARISON LENGTH      UNPREDICTABLE      *
* R8          ADDR OF DEVICE FIELD    UNCHANGED        *
* R9          N/A          UNCHANGED        *
* R10         ADDR OF COMPARISON FIELD UNCHANGED        *
* R11         HCT BASE ADDRESS        UNCHANGED        *
* R12         N/A          UNCHANGED        *
* R13         PCE BASE ADDRESS        UNCHANGED        *
* R14         RETURN ADDRESS        UNCHANGED        *
* R15         ENTRY ADDRESS          0          UNCHANGED        *
*
* PARAMETER LIST:
*
*          NONE
*
* REGISTER USAGE (INTERNAL):
*
* REG          VALUE
*
* R0          N/A
* R1          ADDR OF JQE
*
*
* R2          ADDR OF CRITERION BEING *
*          PROCESSED                  *
* R4-R5       N/A
* R6          N/A
* R7          COMPARISON LENGTH
* R8          ADDR OF DEVICE FIELD
* R9          N/A
* R10         ADDR OF COMPARISON FIELD
* R11         HCT BASE ADDRESS
* R12         N/A
* R13         PCE BASE ADDRESS
* R14         LINKAGE REGISTER
* R15         LINKAGE REGISTER
*
* RETURN CODES (R15 ON EXIT):
*
*          4 - CONTINUE CRITERIA PROCESSING, ACCEPTABLE CONDITION
*          12 - UNACCEPTABLE CONDITION, CRITERIA DO NOT MATCH
*
* OTHER CONSIDERATIONS:
*
*          $SAVE AND $RETURN NOT USED FOR PERFORMANCE REASONS
*
*****
SPACE 1
ENTRY WSTRKGRP          ESTABLISH ENTRY POINT
USING WSTRKGRP,R6       ESTABLISH ADDRESSABILITY
USING PCE,R13           ESTABLISH PCE ADDRESSABILITY
SPACE 1
WSTRKGRP LR R6,R15      SET ADDRESSABILITY
BCTR R7,0              PREPARE LENGTH FOR EXECUTES
LR R15,R10             SET THE JQE FIELD ADDRESS
SL R15,=A(JQETGNUM-JQE) TO OBTAIN THE JQE ADDRESS
LR R1,R10              OBTAIN THE FIELD ADDRESS
TM JQEFLAG5-JQE(R15),JQE5XUSD NUM OF TGS IN EXT AREA...
BNO WSTTGN             NO, GO DO COMPARISON
LH R1,JQETGNUM-JQE(,R15) GET THE OFFSET INTO EXT AREA
AL R1,$JQEEXT          AND OBTAIN THE ADDRESS OF TGN
WSTTGN LA R15,12        ASSUME TG NO. NOT AT THRESHOLD
EX R7,WSTCLC           TG NUMBER AT THRESHOLD...
BLR R14               NO, RETURN INDICATING NO MATCH
LA R15,4              YES, INDICATE MATCH
BR R14               RETURN TO CALLER
SPACE 1
WSTCLC CLC 0(*-*,R1),0(R8) *** EXECUTE ONLY ***
SPACE 1
DROP R6,R13           SUSPEND LOCAL & PCE ADDRESSABILITY

```

## TABLES

```

TITLE 'USER EXTENSION MODULE -- USERPCET - TABLE FOR C
INSTALLATION SECURITY PROCESSOR'
*****
*
*          DEFINE THE PROCESSOR TABLE
*

```

```

*****
SPACE 1
USERPCET $PCETAB TABLE=USER
SCTYPCET $PCETAB NAME=SCTY,DESC='SECURITY',MODULE=HASPXJ00, C
          ENTRYPT=UCTMSCTY,CHAIN=UCTSYPCE,COUNTS=UCTSYNUM, C
          MACRO=$SCYWORK,WORKLEN=SCYLEN,GEN=INIT,DISPTCH=WARM, C
          PCEFLGS=0,FSS=NO,PCEID=(0,UPCESCTY),DCTTAB=**-*
          $PCETAB TABLE=END

          TITLE 'USER EXTENSION MODULE -- USERDTET - TABLE FOR C
                INSTALLATION SECURITY SUBTASK
*****
*
*          DEFINE THE SUBTASK TABLE
*
*****
SPACE 1
USERDTET $DTETAB TABLE=USER
          $DTETAB NAME=SECURITY,ID=UDTESCTY,EPNAME=USCTDTE, C
          EPLOC=UCTMDSCY,HEAD=UCTSYDTE,WORKLEN=SCDLEN, C
          GEN=NO,STAE=NO,SZERO=YES
          $DTETAB TABLE=END

          TITLE 'USER EXTENSION MODULE -- USERTIDT - TABLE FOR C
                INSTALLATION TRACE ID TABLE(S)'
*****
*
*          DEFINE THE TRACE ID TABLE
*
*****
SPACE 1
USERTIDT $TIDTAB TABLE=USER
          $TIDTAB ID=255,FORMAT=TROUT255,NAME=SAFCALL
          $TIDTAB TABLE=END

          TITLE 'USER EXTENTION MODULE -- USERSTWT - TABLE FOR C
                INSTALLATION WORK SELECTION CRITERIA'
*****
*
*          DEFINE THE WORK SELECTION CRITERIA TABLE
*
*****
SPACE 1
USERSTWT $WSTAB TABLE=USER
          $WSTAB NME=TRKGRP,MINLEN=2,ALIAS=TG,FLD=JQETGNUM,CB=JQE, C
          DEVFLD=DCTUSER0,DEVCB=DCT,RTN=WSTRKGRP
          $WSTAB TABLE=END

          TITLE 'USER EXTENSION MODULE -- USEROSTT - TABLE FOR C
                INSTALLATION SCAN TABLE FOR OFFN.STN'
*****
*
*          DEFINE THE OFFLOAD SYSOUT TRANSMITTER OPERAND TABLE
*
*****
SPACE 1
USEROSTT $SCANTAB TABLE=USER
          $SCANTAB NAME=TRKGRP,MINLEN=2,FIELD=(DCTUSER0,2),DSECT=DCT, C
          CONV=NUM,RANGE=(0,32767),CB=PARENT,CALLERS=($SCIRPL, C
          $SCIRPLC,$SCDCMDS,$SCSCMDS)
          $SCANTAB NAME=TG,CONV=ALIAS,SCANTAB=TRKGRP
          $SCANTAB TABLE=END
          EJECT
*****
*
*          LIST THE LITERALS FOR THE HASPXJ00 MODULE
*
*****
SPACE 1
LTORG ,

          TITLE 'USER EXTENSION MODULE -- EPILOG ($MODEND)
$MODEND ,

```

APARNUM	DC	CL7'0ZXXXXX'	APAR NUMBER
	END	,	END OF HASPXJ00



---

## Appendix C. Miscellaneous facilities support

This appendix contains several facilities to allow you to further customize your JES2. The following facilities are included:

- “Generalized JES2 dispatcher support” on page 479
- “Data space usage” on page 479
- “General purpose subtasking facility” on page 480
- Appendix E, “Invoking the security authorization facility (SAF),” on page 487

---

### Generalized JES2 dispatcher support

The JES2 dispatcher is completely generalized, thereby making it easy for you to add processor (PCE) resource queues without source code modification.

There are 64 general resource queues, of which JES2 uses the first 23. The resource queue heads are in their own area pointed to by the \$DRQUES field of the HCT. The HCT also contains the event control fields and the \$READY queue.

You can issue your own \$WAITs and \$POSTs, using equated symbols (or hard-coded values, that is, \$WAIT 54) in the same manner as JES2. You can also use \$\$POST to post JES2 main task resources from subtasks or other address spaces. The cross-system operand (MASPOST=) on the \$POST macro is also supported in this generalized scheme.

---

### Data space usage

**Data Spaces** increase the amount of contiguous data that you can access. JES2 supports the use of data spaces through the \$ARMODE and \$DSPSERV macros. The macros are used to build and access data spaces.

A **data space** is an area of virtual storage that the system creates for a user. It provides the largest possible space for contiguous data, up to two gigabytes. You can have multiple data spaces. A data space **contains data only**.

When a data space is created, the system gives the creating task an **STOKEN** (space token). The STOKEN is an address space and data space identifier. When the STOKEN is built, it is added to an access list.

An access list is a table in which each entry specifies an address space or a data space. Each entry in the access list is pointed to by an access list entry token (ALET). The **ALET** is placed in an access register (AR) to access the data in a data space by qualifying the address contained in a general purpose register.

#### \$ARMODE

Use the **\$ARMODE** macro to turn the AR-mode ON or OFF, or to load the access registers with ALETs. The access registers are loaded using the AR= parameter and either the INIT= keyword or the ALET= keyword.

#### \$DSPSERV

Use the **\$DSPSERV** macro to create or delete a data space. The following parameters are used to manage the data space:

##### NAME

is used to name a data space. This name can help you locate the data space in an IPCS dump.

##### STOKEN

names a register that will contain either the STOKEN that was created for the data space or the STOKEN of a data space you want to delete.

**ALET**

specifies the register or name of the storage area that contains the ALET of the data space to be deleted, or the register or name of the storage area to contain the ALET of a new data space.

**BLOCKS**

specifies the size (in 4K blocks) of the data space.

You can also specify if the data space is fetch protected (**FPROT**) or the owner of the data space (**OWNER**).

For additional information about data space usage, see the following publications:

- [\*z/OS MVS Programming: Extended Addressability Guide\*](#)
- [\*z/OS MVS Programming: Assembler Services Guide\*](#)

## General purpose subtasking facility

---

The general purpose subtasking facility provides an easy way to call a service routine from the JES2 main task. Use this facility to provide services, from the main task, that might:

- Cause an MVS WAIT to occur
- Perform input or output operations
- Perform intensive calculations.

Using this facility increases your system's parallel operations without forcing the service to use a particular processor control element (PCE).

## Using the general purpose subtasking facility

To use the general purpose subtasking facility you must:

- Specify the number of general purpose subtasks that will be available on the SUBTDEF initialization statement. The default number of subtasks is 10.
- Obtain a subtask queue descriptor (\$SQD). Use the \$GETWORK macro to obtain the \$SQD. Initialize the \$SQD with an ID and VERSION.
- Determine what the priority for this subtask should be. The different priorities are:
  - High priority – use for quick, interactive type requests. A service provided to a TSO/E user is an example of a high-priority request.
  - Low priority – calculation intensive or non-critical requests. Purge processing is an example of a low-priority request.
  - Regular priority – most requests.

Use the PRIORITY= parameter of the \$SUBIT macro to indicate this priority.

- Begin subtask processing of the routine by coding a \$SUBIT macro with the appropriate parameters.

When \$SUBIT returns to the calling routine, register 15 contains the return code from the subtask processing. The \$SUBIT macro description includes the meaning of each return code. The \$SQD will contain the contents of registers 0, 1, and 15 from the service you invoked.

---

## Appendix D. Accessing checkpoint control blocks outside the JES2 main task

When writing JES2 exits that:

- run outside the JES2 main task and
- need to access or update checkpoint control blocks

you need to follow the specific coding recommendations in this section as well as those specific "Programming Considerations" described for the JES2 exit that you are implementing.

Before you can access the JQE or CAT control blocks, you need to obtain a checkpoint version (copy of the checkpoint data set) by invoking the \$DSERV macro. Non-main task exits can access checkpoint control blocks like JQEs, CATs in read mode only. You can also access other control blocks such as the BERT, JOE, and WSC.

Consider the following series of calls to obtain the CAT (class attribute table) in read mode and then return it:

1. Use the following \$DSERV call to obtain the checkpoint version:

```
$DSERV FUNC=GET,DSERV=addr-x
```

2. When JES2 returns the DSERV, issue a \$DOGCAT call to obtain the \$CAT from the checkpoint version. (addr-x is the version address from the first \$DSERV call). Specify JOBCLASS= to obtain a specific CAT.

```
$DOGCAT ACTION=(FETCH,READ),CAT=addr-z,DSERV=addr-x,JOBCLASS=y
```

3. When you are finished extracting information from the CAT, use the following action to return the CAT obtained (addr-z is the address from the first \$DOGCAT call):

```
$DOGCAT ACTION=RETURN,CAT=addr-z,DSERV=addr-x
```

4. 4. Now issue a second \$DSERV to return the checkpoint version (addr-x is the address of the DSERV obtained from the first \$DSERV call):

```
$DSERV FUNC=FREE,DSERV=addr-x
```

---

### Typical macro calls

If you have implemented Exit 6 (the JES2 converter exit), Exit 8 (control block read/write), or Exit 12 (spool partitioning allocation - \$STRAK) you are aware that these exits are within the JES2 subtask environment, not the JES2 main task. They are thereby subject to data-set-access restrictions such as the inability to directly access the JES2 checkpoint data set. If you need to access the checkpoint control blocks, for example, when developing an Exit 6 routine to get the details of a particular jobclass, you can only do so by accessing a checkpoint version and reading that copy of the CAT (class attributes table). When issuing \$#JOE, \$DOGCAT, \$DOGJQE, \$DOGWSCQ, \$QJQE, or \$QLOC service calls, you must first access a checkpoint version using a \$DSERV GET call. Use the code examples below to assist coding these macro calls. Then, invoke SAF by issuing the \$SEAS macro specifying the required parameters.

### \$QJQE

Holding the checkpoint version over the \$QJQE loop ensures that the chain of JQEs being processed are not altered while they are being examined. Once you obtain a DSERV, you can call additional services that require a DSERV and be assured that none of the data areas have changed while the checkpoint data is being processed. However, because the number of checkpoint versions is set at the system level, holding

a checkpoint level for an extended period of time can have negative performance implications for other address spaces and processes.

```

USING DSERV,Rx
$DSERV FUNC=GET,          Access a copy of the checkpoint data
  ERRET=xxxxxx
LR   Rx,R1
:
$QJQE ... ,
  DSERV=DSERV,
  LOOP=QLOOP,
  NOMORE=QEND
:
process JQE data
:
J    QLOOP

QEND  $DSERV FUNC=FREE,DSERV=DSERV
DROP  Rx

```

Figure 22. Typical \$DSERV call when obtaining a JQE with \$QJQE

## \$#JOE

Holding the checkpoint version over the \$#JOE loop ensures that the chain of JQEs being processed are not altered while they are being examined. Once you obtain a DSERV, you can call additional services that require a DSERV and be assured that none of the data areas have changed while the checkpoint data is being processed. However, because the number of checkpoint versions is set at the system level, holding a checkpoint level for an extended period of time can have negative performance implications for other address spaces and processes.

```

USING DSERV,Rx
$DSERV FUNC=GET,          Access a copy of the checkpoint data
  ERRET=xxxxxx
LR   Rx,R1
:
$#JOE ... ,
  DSERV=DSERV,
  LOOP=QLOOP,
  NOMORE=QEND
:
process JOE data
:
J    QLOOP

QEND  $DSERV FUNC=FREE,DSERV=DSERV
DROP  Rx

```

Figure 23. Typical \$DSERV call when validating a queue with \$#JOE

## \$DOGBERT

Holding the checkpoint version over the \$DOGBERT fetch-return cycle ensures that BERT being processed are not altered while they are being examined. Once you obtain a DSERV, you can call additional services that require a DSERV and be assured that none of the data areas have changed while the checkpoint data is being processed. However, because the number of checkpoint versions is set at the system level, holding a checkpoint level for an extended period of time can have negative performance implications for other address spaces and processes



```

USING DSERV,Rx

$DSERV FUNC=GET,           Access a copy of the checkpoint data
  ERRET=xxxxxx
LR    Rx,R1
:
$DOGBERT ... ,
  ACTION=(FETCH,READ),
  DSERV=DSERV
:
process BERT
:
$DOGBERT ... ,
  ACTION=RETURN,
  DSERV=DSERV

$DSERV FUNC=FREE,DSERV=DSERV

DROP Rx

```

Figure 24. Typical \$DSERV call when obtaining a BERT data with \$DOGBERT

## \$DOGCAT

Holding the checkpoint version over the \$DOGCAT fetch-return cycle ensures that CAT being processed are not altered while they are being examined. Once you obtain a DSERV, you can call additional services that require a DSERV and be assured that none of the data areas have changed while the checkpoint data is being processed. However, because the number of checkpoint versions is set at the system level, holding a checkpoint level for an extended period of time can have negative performance implications for other address spaces and processes

```

USING DSERV,Rx

$DSERV FUNC=GET,           Access a copy of the checkpoint data
  ERRET=xxxxxx
LR    Rx,R1
:
$DOGCAT ... ,
  ACTION=(FETCH,READ),
  DSERV=DSERV
:
process CAT data
:
$DOGCAT ... ,
  ACTION=RETURN,
  DSERV=DSERV

$DSERV FUNC=FREE,DSERV=DSERV

DROP Rx

```

Figure 25. Typical \$DSERV call when obtaining a CAT with \$DOGCAT

## \$DOGJQE

Holding the checkpoint version over the \$DOGJQE fetch-return cycle ensures that JQE being processed are not altered while they are being examined. Once you obtain a DSERV, you can call additional services that require a DSERV and be assured that none of the data areas have changed while the checkpoint data is being processed. However, because the number of checkpoint versions is set at the system level, holding a checkpoint level for an extended period of time can have negative performance implications for other address spaces and processes

```

USING DSERV,Rx

$DSERV FUNC=GET,           Access a copy of the checkpoint data
  ERRET=xxxxxx
LR    Rx,R1
:
$DOGJQE ... ,
  ACTION=(FRETCH,READ),
  DSERV=DSERV
:
process JQE data
:
$DOGJQE ... ,
  ACTION=RETURN,
  DSERV=DSERV

$DSERV FUNC=FREE,DSERV=DSERV

DROP  Rx

```

Figure 26. Typical \$DSERV call when obtaining an artificial JQE (JQA) with \$DOGJQE

## \$DOGWSCQ

The \$DOGWSCQ when used outside the JES2 main task, cannot access the JES2 checkpoint directly. It must access the checkpoint using a checkpoint version (a copy of the current checkpoint in a data space). A checkpoint version is described by the IAZDSERV data area which must be obtained prior to invoking the \$DOGWSCQ service. The \$DSERV MACRO can be used to obtain an IAZDSERV data area for use by the \$DOGWSCQ service. Processing is typically:

```

USING DSERV,Rx

$DSERV FUNC=GET,           Access a copy of the checkpoint data
  ERRET=xxxxxx
LR    Rx,R1
:
$DOGWSCQ ... ,
  ACTION=(FRETCH,READ),
  DSERV=DSERV
:
process WSC data
:
$DOGWSCQ ... ,
  ACTION=RETURN,
  DSERV=DSERV

$DSERV FUNC=FREE,DSERV=DSERV

DROP  Rx

```

Figure 27. Typical \$DSERV call when obtaining a workload management service class with \$DOGWSCQ

## \$QLOC

When \$QLOC is used outside the JES2 maintask, there is a need to access the checkpoint using a checkpoint version (a copy of the current checkpoint in a data space). You must obtain the IAZDSERV data area, which describes the checkpoint version, prior to invoking the \$QLOC service. Use a \$DSERV GET call to do so. See "Accessing checkpoint control blocks outside the main task" in Appendix D for a typical coding example.

```

USING DSERV,Rx
$DSERV FUNC=GET,          Access a copy of the checkpoint data
    ERRET=xxxxxx
LR    Rx,R1
:
$QLOC ... ,
    JOBNUM=JQEJOBNO_R4,
    DSERV=DSERV
:
process JQE data
:

$DSERV FUNC=FREE,DSERV=DSERV

DROP  Rx

```

*Figure 28. Typical \$DSERV call when obtaining a job queue element with \$QLOC*



---

## Appendix E. Invoking the security authorization facility (SAF)

The security authorization facility (SAF) is the component of MVS that verifies a user's authorization to resources. These resources include commands, data sets, jobs, nodes, printers, card readers, remotes, SYSIN, and SYSOUT. SAF verifies the authorization by routing requests to the Resource Access Control Facility (RACF®) or any functionally equivalent security product. RACF determines if the user has authority to a particular resource.

All JES2 resources have profile names in a specific format. See [z/OS JES2 Initialization and Tuning Guide](#) for examples of profile names.

JES2 invokes SAF when attempting to access a resource on behalf of itself or a user. An installation can enhance SAF verification through:

- Exits 36 and/or 37
- Additional calls to SAF through \$SEAS
- Defining additional resource names and verifying those resources through SAF. The security administrator must then define the required resources to RACF in specific resource classes.

---

### Using \$SEAS to invoke SAF

\$SEAS is the macro JES2 uses as an interface to SAF. To invoke \$SEAS you must:

1. Get a work access verification element (\$WAVE) and place an ID and version in it. The \$WAVE macro maps the work access verification element.
2. Clear the WAVESQD field of the \$WAVE.
3. Copy the list form of the RACROUTE macro into the WAVRACRP field of the \$WAVE. For more information about the RACROUTE macro, see [z/OS MVS Programming: Assembler Services Reference ABE-HSP](#).
4. Issue the modify form of the RACROUTE macro to update the parameter list at WAVRACRP and supply any other parameters the RACROUTE macro requires.

Then, invoke SAF by issuing the \$SEAS macro specifying the required parameters

- ENVIRON=
- FUNCODE=
- REQUEST=
- PRIORITY=
- WAIT= and WAVADDR= In most cases, you place a request to SAF and wait for a response. To do this, code WAIT=YES and WAVADDR=(addr,INIT). When SAF completes your request, register 15 contains the result of the verification as a return code.

[“#unique\\_879/unique\\_879\\_Connect\\_42\\_seasxmp” on page 487](#) illustrates a standard SAF call using \$SEAS.

```
*****
*                                     *
*      Get storage for a WAVE         *
*                                     *
*****

      GETMAIN RU,L=WAVLEN           Get WAVE for $SEAS

      USING WAVE,R3                Establish WAVE addressability
      LR      R3,R1                Set WAVE base address
```

```

L      R10,TOKEN      Set token address of requestor

*****
*
*      Initialize RACF parameter list in the WAVE
*
*****

MVC    WAVRACRP(L'RACROUTL),RACROUTL  Set parameter list

RACROUTE REQUEST=AUTH,WORKA=WAVRRWK,ATTR=UPDATE,
        ENTITY=PDBDSNAM,REQSTOR=SAFAUTH,RELEASE=1.9,
        LOGSTR=LGSYSSAF,RTOKEN=PDBTOKEN,CLASS=SPLCLASS,
        UTOKEN=(R10),MF=(M,WAVRACRP)
C
C
C

*****
*
*      Invoke the $SEAS macro to verify access.
*      Even though CODER=USER is the default, it is shown here
*      for clarity.
*
*****

$SEAS CODER=USER,WAVADDR=(R3),REQUEST=AUTH,FUNCODE=255,
        JOBMASK=(R4)      Make the required SAF call
C

*****
*
*      ENVIRON=, PRIORITY=, and WAIT= defaulted. Values taken are:
*      ENVIRON= -- current assembly environment
*      PRIORITY= -- Regular
*      WAIT= -- YES
*
*****

SAFRETCD B    SAFRETCD(15)      Process return code
SAFRETCD B    SAFOK              +0 - User authorized
        B    SAFHUH            +4 - SAF cannot decide
        B    SAFFAIL          +8 - User not authorized
        .
        .
        .

*****
*
*      RACROUTE macro parameters "REQSTOR" and "LOGSTR" used
*      in the audit record to identify the call.
*
*****

LGSYSSAF DC    AL1(L'LGSYSST)      RACROUTE
LGSYSST  DC    C'USER DATA SET AUTH CALL '  Log string

SAFAUTH  DC    CL8'USERAUTH'      Requestor ID for AUTH
                                           call

*****
*
*      RACROUTE list form for REQUEST=AUTH for data set authorization
*
*****

RACROUTX RACROUTE REQUEST=AUTH,MF=L,RELEASE=1.9,DECOUPL=YES
RACROUTL EQU    RACROUTX,*-RACROUTX  Length of list form

*****
*
*      RACROUTE class name for spool data sets
*
*****

```

SPLCL	DC	AL1(L'SPLCLV)	Length of JESSPOOL class
SPLCLV	DC	C'JESSPOOL'	JESSPOOL class
SPLCLASS	EQU	SPLCL,*-SPLCL,C'X'	JESSPOOL SAF class

There may be times that your routine cannot wait for SAF to complete its processing. In this case, you need to call \$SEAS twice. For the first call, code WAIT=NO and WAVADDR=(addr,INIT). This allows your program to continue its work while SAF starts its processing. A non-zero return code at this point indicates JES2 cannot make the request and you should decide what action to take. To determine when SAF has completed its processing, test the field SQDXECB against ECBPOST. When the result is non-zero, the processing is complete.

If JES2 completes the first call successfully, you must reissue the \$SEAS macro with WAIT=YES and WAVADDR=(addr,NOINIT) to wait for SAF to complete verification. **Do not** clear the WAVESQD before making the second call. [“#unique\\_879/unique\\_879\\_Connect\\_42\\_seasxp2” on page 489](#) illustrates this type of SAF call.

```
*****
*
*      Get storage for a WAVE
*
*****

        GETMAIN RU,L=WAVLEN          Get WAVE for $SEAS

        USING WAVE,R3              Establish WAVE addressability
        LR     R3,R1               Set WAVE base address

        L      R10,TOKEN           Set token address of requestor

*****
*
*      Initialize RACF parameter list in the WAVE
*
*****

        MVC    WAVRACRP(L'RACROUTL),RACROUTL   Set parm list

        RACROUTE REQUEST=AUTH,WORKA=WAVRRWK,ATTR=READ,
                ENTITY=PDBDSNAM,REQSTOR=SAFAUTH,RELEASE=1.9,
                LOGSTR=LGSYSSAF,RTOKEN=PDBTOKEN,CLASS=SPLCLASS,
                UTOKEN=(R10),MF=(M,WAVRACRP)

*****
*
*      Invoke the $SEAS macro to initiate a SAF authorization
*      request
*      Even though CODER=USER is the default, it is shown here
*      for clarity.
*
*****

        $SEAS CODER=USER,WAVADDR=(R3),REQUEST=AUTH,FUNCODE=254,
                WAIT=NO                      Make the required SAF call

*****
*
*      ENVIRON=, and PRIORITY= defaulted. Values taken are:
*      ENVIRON= -- current assembly environment
*      PRIORITY= -- Regular
*
*****

        .
        .                                Perform other work
        .

*****
*
*      Invoke the $SEAS macro to verify access.
*      Even though CODER=USER and WAIT=YES are defaults, they are
```

```

*          shown here for clarity.          *
*                                          *
*****
          $SEAS CODER=USER,WAVADDR=(R3,N0INIT),WAIT=YES          C
                    Make the required SAF call

          B    SAFRETC(15)          Process return code
SAFRETC B    SAFOK          +0 - User authorized
          B    SAFHUH          +4 - SAF cannot decide
          B    SAFFAIL          +8 - User not authorized
          .
          .
*****
*          RACROUTE macro parameters "REQSTOR" and "LOGSTR" used          *
*          in the audit record to identify the call.          *
*          *          *
*****

LGSYSSAF DC    AL1(L'LGSYSST)          RACROUTE
LGSYSST DC    C'READ ACCESS AUTH CALL ' Log string

SAFAUTH DC    CL8'USERAUTH'          Requestor ID for AUTH call

*****
*          *          *
*          RACROUTE list form for REQUEST=AUTH for data set authorization *
*          *          *
*****

RACROUTX RACROUTE REQUEST=AUTH,MF=L,RELEASE=1.9,DECOUPL=YES
RACROUTL EQU  RACROUTX,*-RACROUTX Length of list form

*****
*          *          *
*          RACROUTE class name for spool data sets          *
*          *          *
*****

SPLCL DC    AL1(L'SPLCLV)          Length of JESSPOOL class
SPLCLV DC    C'JESSPOOL'          JESSPOOL class
SPLCLASS EQU  SPLCL,*-SPLCL,C'X' JESSPOOL SAF class

```



## Appendix F. Techniques for writing multi-environment access

When you have a function that is required in both a main task and a user environment exit, you need to know how to package the code. If you are attempting to implement a simple function, the easiest approach is to duplicate your main task exit routine as a new user environment exit; however, keeping the functions in both exits updated could present a maintenance problem. If this is a concern, you need to have only one source file that contains most of the code for the function. Here are some options for maintaining only one copy of your code:

- Write the code to be completely environment neutral (does not use any JES2 services or data areas other than what was passed).
- Invoke a common routine from both exits (through \$CALL or \$SUBIT).
- Use COPY code or MACROs to include common code from a single source.
- Use \$EXIT to invoke common routines.
- The JES2 code uses the second option for most of the code in z/OS V1R7. Services were introduced in z/OS V1R7 to simplify using this method of writing code.

To illustrate the considerations for writing code that can be used from multiple environments, you can start with a trivial exit 2 that looks at the job name and fails any job with a job name of TOM. The first method is to write environment neutral code. See the following example:

	USING JRW,R8	Est JRW addressability
	USING XPL,R10	Est XPL addressability
	SPACE 1	
EXIT002	\$ENTRY BASE=R12	Provide exit routine entry point
	BAKR R14,0	Save callers register
	LR R12,R15	Set local base
	LR R10,R0	Set \$XPL base address
	L R8,X002AREA	Get JRW address
	TM X002COND,X002CONT	This a continuation?
	JO EXITRC0	Yes, just exit
	SPACE 1	
	CLC JRWSTMTL,=CL8'TOM'	Job name set to TOM?
	JNE EXITRC0	No, allow job
	LHI R15,8	Yes, indicate failure
	J EXIT	Return to caller
	SPACE 1	
EXITRC0	LHI R15,0	Set good return code
	SPACE 1	
EXIT	EREG R0,R1	Restore callers R0 and R1
	PR ,	
	SPACE 1	
	DROP R8,R10	Drop JRW, XPL
	SPACE 1	
	LTORG ,	

By loading this code in CSA (using a LOADMOD(USERX002) STORAGE=CSA initialization statement), you can use this exit code for both the main task exit 2 and the user environment exit 52. However, notice that the only JES2 service that is used by this exit is the \$ENTRY statement (which generates inline code). There are also no references to anything other than what was defined in the exit interface and the linkage stack was used to store the registers (valid in the main task because no \$WAITs were issued), not the standard \$SAVE.

To make the exit more interesting, you can make a slight change. Instead of always failing all jobs with a job name of TOM, you can obtain the name of the jobs to fail from a user field in a control block. Because we want to be able to use this from exit 52, you need a CSA data area to store the target job name. You can use CCTUSER1 and CCTUSER2 to store the name of the jobs to fail. The CLC in the middle of the previous example changes to "CLC JRWSTMTL,CCTUSER1." However, the first environmental problem appears. When called as an exit 2, register 11 is the HCT and when called as exit 52, register 11 is the HCCT. Therefore, we need code that can detect what exit you are in and set up the proper register 11.

There are a number of ways to do this, but for this simple exit, you can examine the exit number in the \$XPL to decide how to set up register 11. The result is that the CLC in the first sample is replaced with the following:

```

SPACE 1
CLI    XPLXITID,X002XID      Is this exit 2 calling?
JNE    EXITCLC               No, check the job name
L      R11,$HCCT-HCT(R11)    Set the HCCT address in R11
SPACE 1
USING  HCCT,R11              Est HCCT addressability
SPACE 1
EXITCLC CLC    JRWSTMTL,CCTUSER1 Job have an invalid job name?
JNE    EXITRC0               No, allow job

```

The exit now has some understanding of its environment and is setting registers based on that environment. In addition to register 11, register 13 is environmentally sensitive. If this is exit 2, register 13 is the \$PCE address. If this is exit 52, register 13 is a save area. You can also use this fact to determine who the caller is. If register 13 points to the 4 byte eyecatcher C'PCE ', this is exit 2 called out of the JES2 main task.

Continuing this example, let's assume the requirement is now not to fail the job, but rather, to add a route print JECL card after the job card if the job name matches what is in the CCTUSER1. Adding the card can be done using the new functions provided in z/OS V1R7 to chain an RJCB with the new card after the job card. However, there is a problem. To obtain an RJCB, a call to RGETRJCB must be made. That service assumes that R11 is the HCCT and \$CALL requires that the \$ANVIRON variable is set correctly. As a result, this no longer qualifies as environment neutral code. To address this, the next example uses a common routine called by exits 2 and exit 52.

But first, let us discuss what JES2 environments are and how they are used. At assembly time, there is a global symbol called \$ANVIRON that tells JES2 macros the current run time environment for a section of code. The ENVIRON= keyword on the \$MODULE and \$ENVIRON macros manages the \$ANVIRON symbol. There are seven JES2 environments: DOC, JES2, USER, SUBTASK, FSS, IPCS, or MONITOR. In this discussion we are interested in the JES2 and USER environments. The JES2 environment assumes that the code is running under the JES2 main task in the JES2 address space, that register 13 is the \$PCE, and register 11 is the \$HCT. The USER environment assumes that register 13 is a save area, that register 11 is the \$HCCT, and that we can be running in any address space (under any task). It is important that the \$ANVIRON variable matches the actual environment the code runs in. However, code running in the JES2 address space under the JES2 main task can be considered to be running in the USER environment, because the USER environment covers all tasks in all address space.

The common code we write to solve our new requirement is written using the USER environment (because it can include callers that are in the JES2 environment). This example is comprised of three routines, one for exit 2, one for exit 52 and a common routine to perform the actual requests. These are located in one CSECT (source module) to simplify packaging. The load module must be loaded in CSA because it must be callable from exit 52.

The main task exit 2 routine starts off in the JES2 environment (in order to use \$SAVE, an environmentally sensitive macro), and then uses \$ENVIRON to change to the USER environment (and set R11 to the HCCT). It calls a common service with the same registers 0 and 1 that were passed, upon return restores the JES2 environment, and returns the common routine's return code to the caller.

```

USING  HCT,R11                Est HCT addressability
$ENVIRON SET,ENVIRON=JES2     Set JES2 environment
SPACE 1
EXIT002 $ENTRY BASE=R12,SAVE=YES Define entry and save registers
SPACE 1
$ENVIRON PUSH,ENVIRON=USER,SETR11=YES
SPACE 1
$CALL  EXIT2_52               Call common routine
SPACE 1
$ENVIRON POP,SETR11=YES
SPACE 1
$RETURN RC=(R15)              Return to caller
SPACE 1
DROP   R11,R12                Drop HCT, Local

```

The exit 52 code is very similar to the exit 2 code except that it is running in the USER environment. This results in very different code for the \$SAVE and \$RETURN calls. Notice that the \$ENVIRON calls were not needed around the \$CALL to the common routine because we are already in the user environment.

USING HCCT,R11	Est HCCT addressability
\$ENVIRON SET,ENVIRON=USER	Set USER environment
SPACE 1	
EXIT052 \$ENTRY BASE=R12,SAVE=YES	Define entry and save registers
SPACE 1	
\$CALL EXIT2_52	Call common routine
SPACE 1	
\$RETURN RC=(R15)	Return to caller
SPACE 1	
DROP R11,R12	Drop HCCT, Local

Finally, the common routine checks the job name and inserts the card if needed. More details on how to insert a card and the checks that were made can be found under the description of exit 2 or 52 in [z/OS JES2 Installation Exits](#).

USING JRW,R8	Est JRW addressability
USING XPL,R10	Est XPL addressability
USING HCCT,R11	Est HCCT addressability
USING EXIT2_52,R12	Est HCCT addressability
\$ENVIRON SET,ENVIRON=USER	Set USER environment
SPACE 1	
EXIT2_52 \$SAVE ,	Save callers registers
LR R12,R15	Set local base register
LR R10,R0	Set \$XPL base address
L R8,X002AREA	Get JRW address
TM X002COND,X002CONT+X052SEC	Continuation or second entry?
JNZ EXITRC0	Yes, just exit
SPACE 1	
CLC JRWSTMTL,CCTUSER1	Job name match HCCT value?
JNE EXITRC0	No, don't insert card
SPACE 1	
\$CALL RGETRJCB,PARM=JRW	Get a RJCB for the card
MVC RJCBCARD-RJCB(,R1),EXITCARD	Copy ROUTE PRINT
MVC RJCBRJCB-RJCB(,R1),X002RJCA	Add card to new
ST R1,X002RJCA	card chain
SPACE 1	
EXITRC0 LHI R15,0	Set good return code
SPACE 1	
\$RETURN RC=(R15)	Return to caller
SPACE 1	
EXITCARD DC CL80'/*ROUTE PRINT LOCAL'	
SPACE 1	
DROP R8,R10,R11,R12	Drop JRW, XPL, HCCT, Local
SPACE 1	
LTORG ,	

Looking at this further, because all what the EXIT052 service routine does is to call EXI2\_52, this sample could be simplified to two routines. If the common EXIT2\_52 routine were renamed to EXIT052 and the EXIT002 code simply called EXIT052 (using \$CALL), the same function would have been accomplished with only two routines. How you decide to arrange your routines should be based on which method is clearer to you.

There is one last complication to add to this sample. What if the common routine is more complicated and needs to call a RACF service to verify that the job can run? The problem is that in the JES2 environment (under the main task), the \$SEAS request needs to be subtasked and we need to \$WAIT for the request to complete. However, you cannot use \$WAIT outside the JES2 main task. Or more generally, what if you need to perform some environmentally neutral functions and then call some services that are sensitive to the environment. This was a problem faced by JES2 development in particular when making the \$SCAN facility available outside the main task.

The problem centers around how the \$SAVE and \$RETURN services manage the save areas and what is in register 13 while processing. In the USER environment, the registers are saved on the linkage stack. Because this is a task level data structure that cannot be shared among sub-processes (\$PCEs), you cannot issue a \$WAIT while there is a linkage stack entry. In the JES2 main task, the registers are saved in work areas chained to the PCE pointed to by register 13. Because the save areas are related to the sub-process, \$WAIT can suspend that process and allow another sub-process to run.

To address this problem, code in \$SAVE and \$RETURN was altered to save registers based on the environment of the caller. When the caller is the JES2 main task, the main task save and return services are used. When the caller is outside the JES2 main task, the USER environment save and return services are used. To indicate to the \$SAVE and \$RETURN macros that this behavior is required, a new environment was created called USER ANY. You can set this environment active by specifying ENVIRON=(USER,ANY) on the \$ENVIRON or \$MODULE macros. This is a USER environment which implies that register 11 is the HCCT address. However, because of the way that \$SAVE functions in the USER,ANY environment, register 13 will be a \$PCE address if the caller is the JES2 main task and a \$CSAV save area (user environment save area) if the caller is not the JES2 main task.

There are only a few macros that are sensitive to the new USER ANY environment. These include \$SAVE, \$RETURN, \$ENTRY, \$STORE, \$RESTORE, \$BLDMSG, \$NOTIFY, \$RMSGQUE, and \$SCAN.

Returning to our example, we need to make a RACF call in the common routine for exits 2 and 52. Assuming that the code to set up the parameter list for the RACF call has already been written, the following example just shows how to code the \$SEAS call. It is assumed that a \$ENVIRON SET,ENVIRON=(USER,ANY) was issued before the \$SAVE for the common routine.

```

      CLC      PCEEYE-PCE(,R13),=CL4'PCE'  Running under PCE?
      JE      EXITMT                      Yes, deal with main task
      SPACE 1
$SEAS REQUEST=AUTH,                      Issue
      FUNCODE=$SEADEVA,                  RACROUTE
      WAVADDR=(R2),                      AUTH
      CODER=USER                          call
      J      EXITSAFD                    Process SAF call done
      SPACE 1
EXITMT $ENVIRON PUSH,ENVIRON=JES2,SETR11=YES
      SPACE 1
$SEAS REQUEST=AUTH,                      Issue
      FUNCODE=$SEADEVA,                  RACROUTE
      WAVADDR=(R2),                      AUTH
      CODER=USER                          call
$ENVIRON POP,SETR11=YES                  Restore environment
      SPACE 1
EXITSAFD DS    0H                        Start processing return code

```

Though the 2 \$SEAS calls specify identical parameters, the fact that they are being invoked in different environments causes the code to expand differently, calling 2 separate routines. A \$ENVIRON to set ENVIRON=USER was not needed because the code is already running in the USER environment prior to the first \$SEAS macro invocation.

Another approach to calling services, such as RACF, that cannot be invoked directly from the main task is to use \$SUBIT to subtask the entire routine. In the above example, the exit 2 routine could invoke a common exit 52 routine using the following \$SUBIT call:

```
$SUBIT EXIT2_52,PARM0=(R0),PARM1=(R1),R11=HCCT
```

This approach is especially useful if the old exit 2 function used \$SUBIT to invoke a service. The code in the old exit 2 can be moved to a new routine, updated for the new data areas added in z/OS V1R7 and the \$SUBITed service called directly. Then this new common routine can be \$CALLED from exit 52 and \$SUBITed from exit 2.

Another approach to dealing with having one set of code in multiple exit environments is to use macro or copy code. For simple exits, this is better than making 2 copies of the code. It also allows the use of macro language to determine the JES2 environment instead of a run time check. If you are comfortable writing macro code or you are already using macro language to support multiple versions of the code, writing the needed macro code to support multiple exits would be a logical extension. As long as the JES2 environment is correctly set before invoking your macro or including your copy code, any JES2 macros you invoke expand correctly.

Using the \$EXIT facility to invoke one exit point from another (for example, invoke exit 52 from exit 2) is similar to having a common routine invoked from separate exit routines. However, in this case, your one exit 2 routine invoke all the exit 52 routines. If they can all handle being called out of the main task, this simplifies the maintenance of your exit points. Because the JES2 user environment exit effector runs in the (USER,ANY) environment, the exit 52 code can detect that the caller is from the JES2 main task and,

after setting the correct environment, invoke main task specific services. Just as when a single common routine was called, you must ensure that the \$EXIT macro is also invoked in the proper environment (USER,ANY). Therefore, the result is that your single exit 2 would be as follows:

```

        USING JCT,R10           Est JCT addressability
        USING HCT,R11           Est HCT addressability
        $ENVIRON SET,ENVIRON=JES2 Set JES2 environment
        SPACE 1
EXIT002 $ENTRY BASE=R12,SAVE=YES Define entry and save registers
        LR      R2,R0           Set XPL base
        SPACE 1
        $ENVIRON PUSH,ENVIRON=(USER,ANY),SETR11=YES
        SPACE 1
        ICM     R10,B'1111',X002JCT-XPL(R2) Get JCT address
        JNZ     EXITJCT         Process exit with JCT
        SPACE 1
        $EXIT    52,XPL=(R0),MAXRC=16, Invoke exit 52
                     NOENTER=EXITRET0 Not enabled, exit
        J        EXITRTN        Return to caller
        SPACE 1
EXITJCT $EXIT    52,XPL=(R0),MAXRC=16, Invoke exit 52
                     JOBMASK=JCTXMASK, with exit mask
                     NOENTER=EXITRET0 Not enabled, exit
        J        EXITRTN        Return to caller
        SPACE 1
EXITRET0 LHI     R15,0           Set good return code
        SPACE 1
EXITRTN $ENVIRON POP,SETR11=YES
        SPACE 1
        $RETURN RC=(R15)        Return to caller
        SPACE 1
        DROP    R10,R11,R12     Drop HCT, JCT, Local

```

In this case, the test for the \$JCT is unnecessary because exit 2 is always called with a \$JCT. However, this is not true for all exits (such as an exit 4) so the test was included for completeness. Only one exit 2 can use this technique because a second exit 2 using this method would simply call the same exit 52 routines that the first exit 2 called. Remember, this assumes that all the exit 52 routines can be called from the main task. If any of the exit 52 routines do an MVS wait without checking the environment, serious performance problems or even deadlocks could result.



---

## Appendix G. Accessibility

Accessible publications for this product are offered through [IBM Documentation \(www.ibm.com/docs/en/zos\)](http://www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the [Contact the z/OS team web page \(www.ibm.com/systems/campaignmail/z/zos/contact\\_z\)](http://www.ibm.com/systems/campaignmail/z/zos/contact_z) or use the following mailing address.

IBM Corporation  
Attention: MHVRCFS Reader Comments  
Department H6MA, Building 707  
2455 South Road  
Poughkeepsie, NY 12601-5400  
United States

---

### Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

---

### Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

---

### Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Vol I*

---

### Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Documentation with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that the screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1)

are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The \* symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element \*FILE with dotted decimal number 3 is given the format 3 \\* FILE. Format 3\* FILE indicates that syntax element FILE repeats. Format 3\* \\* FILE indicates that syntax element \* FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines 5.1\*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you must refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

#### **? indicates an optional syntax element**

The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that the syntax elements NOTIFY and UPDATE are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

#### **! indicates a default syntax element**

The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In the example, if you include the FILE keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE (KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

#### **\* indicates an optional syntax element that is repeatable**

The asterisk or glyph (\*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the \* symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1\* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3\* , 3 HOST, 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

#### **Notes:**



1. If a dotted decimal number has an asterisk (\*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.
3. The \* symbol is equivalent to a loopback line in a railroad syntax diagram.

**+ indicates a syntax element that must be included**

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the \* symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the \* symbol, is equivalent to a loopback line in a railroad syntax diagram.



## Notices

---

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing  
IBM Corporation  
North Castle Drive, MD-NC119  
Armonk, NY 10504-1785  
United States of America*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan Ltd.  
19-21, Nihonbashi-Hakozakicho, Chuo-ku  
Tokyo 103-8510, Japan*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

*IBM Corporation  
Site Counsel  
2455 South Road*

*Poughkeepsie, NY 12601-5400*  
*USA*

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

### **COPYRIGHT LICENSE:**

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

## **Terms and conditions for product documentation**

---

Permissions for the use of these publications are granted subject to the following terms and conditions.

### **Applicability**

These terms and conditions are in addition to any terms of use for the IBM website.

### **Personal use**

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

### **Commercial use**

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

## Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

## IBM Online Privacy Statement

---

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at [ibm.com/privacy](http://ibm.com/privacy) and IBM's Online Privacy Statement at [ibm.com/privacy/details](http://ibm.com/privacy/details) in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at [ibm.com/software/info/product-privacy](http://ibm.com/software/info/product-privacy).

## Policy for unsupported hardware

---

Various z/OS elements, such as DFSMSdfp, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

## Minimum supported hardware

---

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: [IBM Lifecycle Support for z/OS \(www.ibm.com/software/support/systemsz/lifecycle\)](http://www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

## Programming Interface Information

---

This publication documents information that is NOT intended to be used as programming Interfaces of JES2.

## Trademarks

---

IBM, the IBM logo, and [ibm.com](http://ibm.com) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at [Copyright and Trademark information \(www.ibm.com/legal/copytrade.shtml\)](http://www.ibm.com/legal/copytrade.shtml).

---

# Index

## Special Characters

- [\\$#ADD macro 20](#)
- [\\$#ALCHK macro 22](#)
- [\\$#BLD macro 23](#)
- [\\$#BUSY macro 24](#)
- [\\$#CAN macro 26](#)
- [\\$#CHK macro 27](#)
- [\\$#DISPRO macro 28](#)
- [\\$#GET macro 29](#)
- [\\$#GETHDJ macro 31](#)
- [\\$#JOE macro 32](#)
- [\\$#JWEL macro](#)
  - [manage work JOEs 35](#)
- [\\$#MOD macro 40](#)
- [\\$#POST macro 40](#)
- [\\$#PUT macro 41](#)
- [\\$#REM macro 42](#)
- [\\$#REP macro 44](#)
- [\\$#TJEV macro 45](#)
- [\\$\\$POST macro 15](#)
- [\\$\\$WTO macro 18](#)
- [\\$\\$WTOR macro 20](#)
- [\\$ACTIVE macro 46](#)
- [\\$ALESERV macro 47](#)
- [\\$ALLOC macro 47](#)
- [\\$AMODE macro 49](#)
- [\\$ARMODE macro 50](#)
- [\\$BERTTAB macro 51](#)
- [\\$BFRBLD macro 53](#)
- [\\$BLDMSG macro 53](#)
- [\\$BLDQC macro 62](#)
- [\\$BLDTGB macro 63](#)
- [\\$BPXCALL macro 63](#)
- [\\$CALL macro 64](#)
- [\\$CBIO macro 67](#)
- [\\$CFSEL macro 73](#)
- [\\$CHECK macro 79](#)
- [\\$CKPT macro 81](#)
- [\\$CPOOL macro 82](#)
- [\\$CWTO macro 89](#)
- [\\$DCBDYN macro 92](#)
- [\\$DCTDYN macro 92](#)
- [\\$DCTTAB macro 93](#)
- [\\$DEST macro 100](#)
- [\\$DESTDYN macro 104](#)
- [\\$DESTID macro 102](#)
- [\\$DILBERT macro 106](#)
- [\\$DISTERR macro 110](#)
- [\\$DOGBERT macro 112](#)
- [\\$DOGCAT macro 119](#)
- [\\$DOGJOE macro 126](#)
- [\\$DOGJQE macro 132](#)
- [\\$DOGWSCQ macro 137](#)
- [\\$DOM macro 140](#)
- [\\$DORMANT macro 141](#)
- [\\$DSERV macro 141](#)
- [\\$DSPSERV macro 143](#)
- [\\$DTEDYN macro 147](#)
- [\\$DTETAB macro 149](#)
- [\\$DVIDBLD macro 152](#)
- [\\$ENTRY macro 153](#)
- [\\$ENVIRON macro 156](#)
- [\\$ERROR macro 158](#)
- [\\$ESTAE macro 161](#)
- [\\$EXCP macro 163](#)
- [\\$EXIT macro 164](#)
- [\\$EXTP macro 166](#)
- [\\$FIFOBLK macro 167, 168](#)
- [\\$FIFOENQ macro 169](#)
- [\\$FIFOGTQ macro 170](#)
- [\\$FRECEL macro 171](#)
- [\\$FRECEMB macro 172](#)
- [\\$FREEBUF macro 172](#)
- [\\$FREJLOK macro 173](#)
- [\\$FRELOK macro 175](#)
- [\\$FREMAIN macro 177](#)
- [\\$FREMLOK macro 176](#)
- [\\$FREQC macro 180](#)
- [\\$FRETBUF macro 181](#)
- [\\$FREUCBS macro 181](#)
- [\\$FREUNIT macro 182](#)
- [\\$FSILINK macro 182](#)
- [\\$GETABLE macro 183](#)
- [\\$GETADDR macro 184](#)
- [\\$GETASCB macro 187](#)
- [\\$GETBLK macro 188](#)
- [\\$GETBUF macro 189](#)
- [\\$GETCEL macro 191](#)
- [\\$GETCMB 193](#)
- [\\$GETHP macro 194](#)
- [\\$GETJLOK macro 195](#)
- [\\$GETLOK macro 197](#)
- [\\$GETMAIN macro 199](#)
- [\\$GETMLOK macro 198](#)
- [\\$GETQC macro 203](#)
- [\\$GETRTN macro 204](#)
- [\\$GETSMFB macro 205](#)
- [\\$GETTBUF macro 206](#)
- [\\$GETUCBS macro 207](#)
- [\\$GETUNIT macro 208](#)
- [\\$GETWORK 209](#)
- [\\$IOERROR macro 210](#)
- [\\$IOTBLD macro 210](#)
- [\\$JBIDBLD macro 212](#)
- [\\$JCAN macro 213](#)
- [\\$JCORBLD macro 215](#)
- [\\$JCT extension 10](#)
- [\\$JCTX macro extension service](#)
  - [JECL statement 11](#)
  - [spool compatibility 12](#)
- [\\$JCTXADD macro 215](#)
- [\\$JCTXEXP macro 220](#)
- [\\$JCTXGET macro 223](#)

[\\$JCTXREM macro 226](#)  
[\\$JQEJNUM macro 229](#)  
[\\$JQESERV macro 230](#)  
[\\$LOGMSG macro 234](#)  
[\\$MID macro 236](#)  
[\\$MODCHK macro 236](#)  
[\\$MODELET macro 240](#)  
[\\$MODEND macro 241](#)  
[\\$MODLOAD macro 241](#)  
[\\$MODULE macro 244](#)  
[\\$MSG macro 262](#)  
[\\$MVCL macro 263](#)  
[\\$NATGET macro 264](#)  
[\\$NHDDADD macro 266](#)  
[\\$NHDEXP macro 268](#)  
[\\$NHDGET macro 269](#)  
[\\$NHDREM macro 270](#)  
[\\$NHDXMT macro 272](#)  
[\\$NITSYNC macro 273, \[275\]\(#\)](#)  
[\\$NOTIFY macro 275](#)  
[\\$PAIR macro 277](#)  
[\\$PATCHSP macro 278](#)  
[\\$PBLOCK macro 279](#)  
[\\$PCEDYN macro 280](#)  
[\\$PCETAB macro 281](#)  
[\\$PCETERM macro 287](#)  
[\\$PDBBLD macro 288](#)  
[\\$PDBFIND macro 289](#)  
[\\$PGSRVC macro 290](#)  
[\\$POST macro 291](#)  
[\\$POSTQ macro 295](#)  
[\\$POSTXEQ macro 295](#)  
[\\$PRPUT macro 296](#)  
[\\$PURGE macro 297](#)  
[\\$PUTABLE macro 297](#)  
[\\$QADD macro 299](#)  
[\\$QBUSY macro 300](#)  
[\\$QCTGEN macro 302](#)  
[\\$QGET macro 302](#)  
[\\$QJIX macro 304](#)  
[\\$QJQE macro 306](#)  
[\\$QLOC macro 309](#)  
[\\$QLOCNXT macro 310](#)  
[\\$QMOD macro 311](#)  
[\\$QPUT macro 313](#)  
[\\$QREM macro 314](#)  
[\\$QSUSE macro 314](#)  
[\\$QUESMFB macro 316](#)  
[\\$QUEUE macro 316](#)  
[\\$RDIRTAB macro 317](#)  
[\\$REPLYV macro 318](#)  
[\\$RESTORE macro 319](#)  
[\\$RETABLE macro 320](#)  
[\\$RETBK macro 321](#)  
[\\$RETSAVE macro 322](#)  
[\\$RETURN macro 322](#)  
[\\$RETNWORK macro 324](#)  
[\\$RMSGQUE macro 324](#)  
[\\$RUSE macro 326](#)  
[\\$SAVE macro 327](#)  
[\\$SCAN facility \[334\]\(#\), \[447\]\(#\)](#)  
[\\$SCAN macro \[329\]\(#\)](#)  
[\\$SCAN table \[448\]\(#\)](#)  
[\\$SCAN table examples \[455\]\(#\)](#)

[\\$SCANB macro 334](#)  
[\\$SCANCOM macro 336](#)  
[\\$SCAND macro 336](#)  
[\\$SCANDIA macro 340](#)  
[\\$SCANTAB macro 341](#)  
[\\$SDUMP macro 355](#)  
[\\$SEAS macro 356](#)  
[\\$SEPPDIR macro 361](#)  
[\\$SETAFF macro 361](#)  
[\\$SETIDAW macro 364](#)  
[\\$SETRP macro 365](#)  
[\\$SJBFINDD macro 365](#)  
[\\$SJBLOCK macro 367](#)  
[\\$SJBRO macro 368](#)  
[\\$SSIBEGN macro 369](#)  
[\\$SSIEND macro 370](#)  
[\\$STCK macro 370](#)  
[\\$STIMER macro 371](#)  
[\\$STMTLOG macro 372](#)  
[\\$STMTTAB macro 373](#)  
[\\$STORE macro 377](#)  
[\\$SUBIT macro 378](#)  
[\\$SYMREC macro 380](#)  
[\\$SYMTAB macro 381](#)  
[\\$TIDTAB macro 385](#)  
[\\$TOKENSR macro 386](#)  
[\\$TRACE macro 388](#)  
[\\$TRACK macro 391](#)  
[\\$TTIMER macro 392](#)  
[\\$USERCBS \[262\]\(#\)](#)  
[\\$VERIFY macro 393](#)  
[\\$VERTAB macro 396](#)  
[\\$VFL macro 398](#)  
[\\$WAIT macro 399](#)  
[\\$WSSETUP macro 402](#)  
[\\$WSTAB macro 403](#)  
[\\$WTO macro - JES2 environment \[411\]\(#\)](#)  
[\\$WTO macro - User and subtask environment \[417\]\(#\)](#)  
[\\$XECBSRV macro 420](#)  
[\\$XMPOST macro \[421\]\(#\)](#)

## A

access  
     multi-address space [50](#)  
 access list  
     definition [479](#)  
 access register register, addressing [479](#)  
 accessibility  
     contact IBM [497](#)  
     features [497](#)  
 acquire a buffer  
     \$GETBUF macro [189](#)  
 acquire a DCT  
     \$GETUNIT macro [208](#)  
 acquire a lock  
     \$FREJLOK macro [173](#)  
     \$GETJLOK macro [195](#)  
     \$GETLOK macro [197](#)  
 acquire a track address  
     \$TRACK [391](#)  
 acquire SMF buffer  
     \$GETSMFB macro [205](#)  
 acquire storage



- acquire storage (*continued*)
  - \$GETMAIN macro [199](#)
- acquire storage area cell
  - \$GETCEL macro [191](#)
- Add a table
  - \$PUTABLE macro [297](#)
- add an element to a FIFO queue
  - \$FIFOENQ macro [169](#)
- add job queue element
  - \$QADD macro [299](#)
- add work JOE to JOT
  - \$#ADD macro [20](#)
- add work JOES
  - \$#JOE macro [35](#)
- address space
  - getting the ASCB [187](#)
  - posting a task [421](#)
  - storage dump (\$SDUMP macro) [355](#)
- affinity
  - \$SETAFF macro [361](#)
- ALET address list entry token
  - definition [479](#)
- ALET service
  - \$ALESERV macro [47](#)
- allocate a job number
  - \$QJIX macro [304](#)
- allocate a unit record device
  - \$ALLOC macro [47](#)
- ARMODE
  - multi-address space access [50](#)
- assembler instruction
  - SYSARM [248](#)
- assembly environment
  - setting [156](#)
- assign message id
  - \$MID macro [236](#)
- assistive technologies [497](#)
- attach a JES2 DESTID
  - \$DESTDYN macro [104](#)
- attach or delete a JES2 PCE
  - \$PCEDYN macro [280](#)

## B

- basic notation
  - macro [5](#)
- begin SSI function
  - \$SSIBEGN macro [369](#)
- BERT table entries
  - \$BERTTAB macro [51](#)
- BERTTAB
  - BERT table entries [51](#)
- block letter
  - generating, \$PBLOCK macro [279](#)
- buffer
  - acquiring (\$GETBUF) [189](#)
  - acquiring (\$GETSMFB macro) [205](#)
  - freeing (\$FREEBUF) [172](#)
  - prefix
    - building (\$BFRBLD macro) [53](#)
    - queueing (\$QUESMFB macro) [316](#)
- buffer obtaining [205](#)
- build a buffer prefix
  - \$BFRBLD macro [53](#)

- build a device name
  - \$DVIDBLD macro [152](#)
- build a JES2 message
  - \$BLDMSG macro [53](#)
- build a job correlator [215](#)
- build a job ID
  - \$JBIDBLD macro [212](#)
- build a Pddb
  - \$PDBBLD macro [288](#)
- build an IOT
  - \$IOTBLD macro [210](#)
- build and map a DTE table
  - \$DTETAB macro [149](#)
- build artificial JOE
  - using \$DOGJOE macro [126](#)
- build artificial JOE
  - using \$DOGJQE macro [132](#)
- build backup storage for a scan
  - \$SCANB [334](#)
- build quick cell pool
  - \$BLDQC macro [62](#)
- build the verification table
  - \$VERTAB macro [396](#)

## C

- call a \$SCAN facility comment service
  - \$SCANCOM macro [336](#)
- call a \$SCAN facility display service routine
  - \$SCAND macro [336](#)
- call a subroutine
  - \$CALL macro [64](#)
- call dynamic DTE service routine
  - \$DTEDYN macro [147](#)
- call omvs services
  - \$BPXCALL macro [63](#)
- call quick cell build/
  - extend
    - \$BLDQC macro [62](#)
- call the \$STCK service routine
  - \$STCK macro [370](#)
- call the \$VERIFY service routine
  - \$VERIFY macro [393](#)
- call the dynamic DCB service routine
  - \$DCBDYN macro [92](#)
- call the dynamic DCT service routine
  - \$DCTDYN macro [92](#)
- call the MODCHECK verification routine
  - \$MODCHK macro [236](#)
- call the MODELET service routine
  - \$MODELET macro [240](#)
- call the MODLOAD service routine
  - \$MODLOAD macro [241](#)
- cancel
  - job
    - \$JCAN macro [213](#)
- cancel work item
  - \$#CAN macro [26](#)
- card Scan table entry
  - \$STMTTAB [373](#)
- catastrophic error
  - indicating [158](#)
  - indicating (\$DISTERR) [110](#)
- cell pool

- cell pool (*continued*)
  - \$CPOOL macro [82](#)
  - get high private storage
    - \$GETHP macro [194](#)
  - managing [82](#)
- channel program
  - executing (\$EXCP) [163](#)
- characteristic of JOEs
  - formatting [23](#)
  - remove a pair from the JOT [42](#)
  - replace a characteristics JOE [44](#)
  - replace a work JOE [44](#)
- check checkpoint write completion
  - \$CHECK macro [79](#)
- CMB (console message buffer)
  - freeing (\$FRECMB macro) [172](#)
- coded value operand [6](#)
- command
  - syntax diagrams [2](#)
- command processor
  - write to operator
    - \$CWTO macro [89](#)
- console
  - message buffer
    - acquiring [193](#)
- contact
  - z/OS [497](#)
- control block address
  - \$GETASCB macro [184](#)
- control block I/O
  - \$CBIO macro [67](#)
- convert destination
  - \$DEST macro [100](#)
- create a name/token pair
  - \$TOKENSR macro [386](#)
- create a scan table [341](#)
- create a user PDIR
  - \$SEPPDIR macro [361](#)
- create separator page
  - \$PRPUT macro [296](#)
- creating a symptom record
  - \$SYMREC macro [380](#)
  - \$SYMTAB macro [381](#)
- cross memory service
  - lock [176](#), [197](#), [198](#)
- CSA storage
  - freeing (\$FRECEL macro) [171](#)
- current PCE
  - obtain work area [209](#)

## D

- data space
  - definition
    - using [479](#)
  - managed by JES2 [143](#)
- data space service
  - \$DSPSERV macro [143](#)
- DCT (device control table)
  - acquiring (\$GETUNIT macro) [208](#)
- DCT table map
  - \$DCTTAB macro [93](#)
- deallocate a job number [304](#)
- define a quick cell control table

- define a quick cell control table (*continued*)
  - \$QCTGEN macro [302](#)
- define a table pair
  - \$PAIR macro [277](#)
- delete operator message
  - \$DOM macro [140](#)
- deliver CAT
  - using \$DOGCAT macro [119](#)
- deliver or get artificial JOE
  - \$DOGJOE macro [126](#)
- deliver or get artificial JQE
  - \$DOGJQE macro [132](#)
- deliver or get BERT data
  - \$DOGBERT macro [112](#)
- deliver or get CAT
  - \$DOGCAT macro [119](#)
- dequeue an entire FIFO queue
  - \$FIFOGTQ macro [170](#)
- destination
  - attach a destination ID [104](#)
  - attaching (\$DESTDYN macro) [104](#)
  - converting (\$DEST macro) [100](#)
- diagnostic message service
  - \$SCANDIA macro [340](#)
- direct access
  - acquiring a track address [391](#)
  - returning space, \$PURGE macro [297](#)
- directing command response
  - to another console
    - \$RDIRTAB macro [317](#)
- do it later BERT services
  - \$DILBERT macro [106](#)
- DSPSERV macro
  - data space management [143](#)
- DTEDYN macro
  - call dynamic DTE service routine [147](#)
- dump storage
  - \$SDUMP macro [355](#)
- dynamic service routine
  - DTE, \$DTEDYN macro [147](#)

## E

- end of module
  - generating, \$MODEND macro [241](#)
- end SSI function
  - \$SSIEND macro [370](#)
- environment
  - of a routine [204](#)
- error
  - disastrous indication [110](#)
- error recovery environment
  - \$ESTAE [161](#)
- establish USING on a register
  - \$RUSE macro [326](#)
- event
  - posted complete [15](#)
  - posting, \$POST macro [291](#)
- example
  - \$SCAN table [455](#)
- execute JES2 channel program
  - \$EXCP macro [163](#)
- execution PCE
  - wake up [295](#)

- exit 0
  - locates scan table [455](#)
- exit 19
  - locates scan table [455](#)
- EXPLICIT= macro parameter
  - \$DESTID macro [102](#)
- EXPLICIT= parameter [102](#)
- extend quick cell pool
  - \$BLDQC macro [62](#)
- extend the \$JCT
  - \$JCTXADD macro [215](#)
  - \$JCTXEXP macro [220](#)

## F

- feedback [xxv](#)
- FIFO (first-in first-out) queue
  - maintaining, \$QUEUE macro [316](#)
- FIFO queue
  - add an element (\$FIFOBLK macro) [169](#)
  - dequeue an entire queue (\$FIFOQTQ macro) [170](#)
  - manage blocking (\$FIFOBLK) [167](#)
  - remove an element (\$FIFOBLK macro) [168](#)
- find and validate queue
  - \$#JOE macro [32](#)
- find work JOEs
  - \$#JOE macro [35](#)
- format a job number [304](#)
- format the JOEs
  - \$#BLD macro [23](#)
- free a CMB
  - \$FRECMB macro [172](#)
- free a DESRV
  - \$DSERV macro [141](#)
- free CMS or job lock
  - \$FRELOK macro [175](#)
- free CSA cell
  - \$FRECEL macro [171](#)
- free quick cell
  - \$FREQC macro [180](#)
- free TCP cell
  - \$FRETBUF macro [181](#)
- freeing storage
  - \$FREMAIN macro [177](#)
  - UCB parameter list (\$FREUCBS macro) [181](#)

## G

- general purpose subtasking facility facility, general purpose
  - subtasking
    - benefit [480](#)
    - using
      - \$SQD [480](#)
      - \$SUBIT [480](#)
    - when to use [480](#)
- generate a \$REPLYV table entry
  - \$REPLYV macro [318](#)
- generate a block letter
  - \$PBLOCK macro [279](#)
- generate end of module
  - \$MODEND macro [241](#)
- generate patch space
  - \$PATCHSP macro [278](#)

- get a HASP/USER table entry
  - \$GETABLE macro [183](#)
- get a storage cell
  - \$GETBLK macro [188](#)
- get address of a routine
  - \$GETRTN macro [204](#)
- get CAT
  - using \$DOGCAT macro [119](#)
- get console message buffer
  - \$GETCMB [193](#)
- get control block address
  - \$GETADDR macro [184](#)
- get held JOE
  - \$#GETHDJ macro [31](#)
- get network header section
  - \$NHDGET macro [269](#)
- get quick cell
  - \$GETQC macro [203](#)
- get TCP cell
  - \$GETTBUF macro [206](#)

## H

- home ASCB
  - retrieving, \$GETASCB macro [187](#)

## I

- I/O error
  - logging, \$IOERROR macro [210](#)
- IDAW (indirect data access word) [364](#)
- identify an exit [153](#)
- implementing
  - \$SCAN table [448](#)
- indicate a catastrophic error
  - \$ERROR macro [158](#)
- indicate a disastrous error
  - \$DISTERR macro [110](#)
- initiate remote terminal
  - I/O
    - \$EXTP macro [166](#)
- installation-defined section
  - adds to an NJE data area
    - \$NHDADD macro [266](#)
  - expand an NJE data area
    - \$NHDEXP macro [268](#)
  - removes from a NJE data area
    - \$NHDREM macro [270](#)
- interface for XECB service
  - \$XECBSRV macro [420](#)
- interval timer
  - setting (\$\$TIMER macro) [371](#)
  - testing (\$TTIMER macro) [392](#)
- IOT build [210](#)

## J

- job
  - lock [197](#)
  - lock freeing (\$FRELOK) [175](#)
- job cancellation
  - \$IOTBLD macro [210](#)
  - \$JCAN macro [213](#)

job correlator macro [215](#)

job ID

    \$JBIDBLD macro [212](#)

job number service

    allocate job number [304](#)

    deallocate job number [304](#)

Job Output Services

    \$#GET macro searching [29](#)

    returning one (\$#PUT macro) [41](#)

JOE busy system indicator

    set [24](#)

JQE busy system indicator

    set [300](#)

## K

keyboard

    navigation [497](#)

    PF keys [497](#)

    shortcut keys [497](#)

## L

link the functional subsystem interface

    \$FSILINK macro [182](#)

locate a \$JCT extension

    \$JCTXGET macro [223](#)

locate a Pddb

    \$PDBFIND macro [289](#)

locate SJB

    \$SJBFINd macro [365](#)

locating a JQE

    \$QLOC macro [309](#)

locating an NAT element

    \$NATGET macro [264](#)

lock

    cms

        acquiring (\$GETLOK macro) [197](#)

        acquiring (\$GETMLOK macro) [198](#)

        releasing (\$FREMLOK macro) [176](#)

    freeing CMS or job lock [175](#)

    job

        acquiring (\$FREJLOK macro) [173](#)

        acquiring (\$GETJLOK macro) [195](#)

        acquiring (\$GETLOK macro) [197](#)

    local

        acquiring (\$GETMLOK macro) [198](#)

        releasing (\$FREMLOK macro) [176](#)

lock the SJB

    \$SJBLOCK macro [367](#)

log a job-related message

    \$LOGMSG macro [234](#)

log an initialization statement

    \$STMTLOG macro [372](#)

log I/O error

    \$IOERROR macro [210](#)

## M

macro

    coded value operand [6](#)

    description [8](#)

    metasymbol [7](#)

macro (*continued*)

    notation [5](#)

    operand representation [5](#)

    operands with value mnemonics [6](#)

    register notation [8](#)

    register stability [8](#)

    value mnemonics

        description [6](#)

macro basic notation [5](#)

macro keyword specification [6](#)

macro parameter [2](#)

macro selection table [8](#)

macro service

    aid to recovery processing [1](#)

    coding aid service [1](#)

    console service [1](#)

    debug service [1](#)

    direct-access space service [1](#)

    error service [1](#)

    general storage management [1](#)

    how provided [1](#)

    initialization service [1](#)

    input/output service [1](#)

    installation exit service [1](#)

    job output service [1](#)

    job queue service [1](#)

    parameters passed [2](#)

    print/punch output service [1](#)

    synchronization service [1](#)

    system management facilities service [1](#)

    table service [1](#)

    time service [1](#)

    unit service [1](#)

    virtual page service [1](#)

    work area management [1](#)

maintain a FIFO queue

    \$QUEUE macro [316](#)

manage blocking of a FIFO queue

    \$FIFOBLOCK macro [167](#)

manage storage cell

    in high private

        \$GETHP macro [194](#)

manage the thread JOE exclusion vector

    \$#TJEV macro [45](#)

manager work JOEs [35](#)

managing cell pools [82](#)

map

    BERT table entries [51](#)

map a PCE table entry

    \$PCETAB macro [281](#)

map and generate work selection table entry

    \$WSTAB macro [403](#)

map trace id table

    \$TIDTAB macro [385](#)

master control table (MCT)

    scanning [448](#)

MCT [448](#)

message id

    \$MID macro [236](#)

message service

    using \$SCANDIA [340](#)

metasymbol [7](#)

modify the JQE

    \$QMOD macro [311](#)

move storage  
    \$MVCL macro [263](#)  
move work JOE  
    \$#MOD macro [40](#)  
multi-address space access  
    \$ARMODE macro [50](#)

## N

navigation  
    keyboard [497](#)  
NJE data area  
    add to  
        data set header [266](#)  
        job header [266](#)  
        job trailer [266](#)  
    expand to  
        job header or data set header [268](#)  
NJE subdevice rolling trace  
    \$NJETRC macro [275](#)  
nodes attached table  
    locating an element [264](#)

## O

obtain a spool record  
    \$#ALCHK macro [22](#)  
    record  
        obtaining (\$#ALCHK macro) [22](#)  
obtain a UCB address  
    \$GETUCBS macro [207](#)  
obtain a work area  
    \$GETWORK [209](#)  
obtain address of JOE queue head  
    \$QJQE macro [306](#)  
obtain job queue element  
    \$QGET macro [302](#)  
obtain save area  
    \$SAVE macro [327](#)  
operand  
    coded value [6](#)  
    keyword [4](#)  
    optional [4](#)  
    positional [4](#)  
    required [4](#)  
    type [4](#)  
operator message  
    area  
        writing, \$MSG macro [262](#)  
        deleting, \$DOM macro [140](#)

## P

packed parameter [4](#)  
page fix  
    \$PGSRVC macro [290](#)  
page free  
    \$PGSRVC macro [290](#)  
page release  
    \$PGSRVC macro [290](#)  
parameter  
    specifying [4](#)  
parameter specification [4](#)

parameter type  
    address [4](#)  
    value [4](#)  
patch space  
    generating, \$PATCHSP macro [278](#)  
PCE (processor control element)  
    \$PCETERM macro [287](#)  
    obtain work area [209](#)  
    saving registers [377](#)  
    termination, \$PCETERM macro [287](#)  
PCE table  
    generating entries, \$PCETAB macro [281](#)  
PDIR (peripheral data information record) [361](#)  
peripheral data definition block  
    \$PDBFIND macro [289](#)  
post a JES2 event complete  
    \$\$POST macro [15](#)  
post a task in another address space  
    \$XMPOST macro [421](#)  
post an event complete  
    \$POST macro [291](#)  
post event  
    \$POST macro [291](#)  
post output device processor  
    \$#POST macro [40](#)  
post resource  
    \$POST macro [291](#)  
posting  
    task [421](#)  
primary ASCB  
    retrieving, \$GETASCB macro [187](#)  
process checkpoint spool  
    I/O  
        \$#CHK macro [27](#)  
process JOE disposition  
    \$#DISPRO macro [28](#)  
processor  
    specify as active (\$ACTIVE) [46](#)  
provide a MIT  
    \$MODULE macro [244](#)  
provide an entry point  
    \$ENTRY macro [153](#)  
provide an exit point  
    \$EXIT macro [164](#)  
purge work JOEs  
    \$#JOE macro [35](#)

## Q

queue  
    finding (\$#JOE macro) [32](#)  
    maintaining, \$QUEUE macro [316](#)  
    shared  
        synchronizing the use (\$QSUSE macro) [314](#)  
    validating (\$#JOE macro) [32](#)  
queue a SMF buffer  
    \$QUESMFB macro [316](#)  
queue message to JES2  
    \$RMSGQUE macro [324](#)  
queue track group blocks  
    \$BLDTGB macro [63](#)  
queueing  
    initiate subtask queueing [378](#)  
queueing a subtask

queueing a subtask (*continued*)

\$SUBIT macro [378](#)

quick post facility

\$POSTQ macro [295](#)

## R

recursive scan [447](#)

register

notation in a macro [8](#)

restoring (\$RESTORE macro) [319](#)

saving (\$SAVE macro) [327](#)

stability macro [8](#)

storing (\$STORE macro) [377](#)

register notation

macro [8](#)

register stability

macro [8](#)

release a DCT

\$FREUNIT macro [182](#)

release storage

\$FREMAIN macro [177](#)

releasing (\$FREUNIT macro) [182](#)

remote terminal

initiate I/O (\$EXTP macro)

[166](#)

remove a \$JCT extension

\$JCTXADD macro [226](#)

remove a JOE

\$QREM macro [314](#)

Remove a table

\$RETABLE macro [320](#)

remove an element from a FIFO queue

\$FIFOSEQ macro [168](#)

remove JOE pair from JOT

\$#REM macro [42](#)

replace a characteristics JOE

\$#REP macro [44](#)

replace a work JOE

\$#REP macro [44](#)

requeue the SJB

\$SJBRQ macro [368](#)

resource

posting, \$POST macro [291](#)

restore register

\$RESTORE macro [319](#)

restore registers

\$RETURN macro [322](#)

retrieve the ASCB

\$GETASCB macro [187](#)

return a JES2 buffer [172](#)

return a JOE

\$#PUT macro [41](#)

unfinished [41](#)

return a JOE

\$QPUT macro [313](#)

return artificial JOE

using \$DOGJOE macro [126](#)

return artificial JOE

using \$DOGJOE macro [132](#)

return direct access space

\$PURGE macro [297](#)

return save area

\$RETSAVE macro [322](#)

return storage cell to pool

\$RETLK macro [321](#)

return to the caller

\$RETURN macro [322](#)

return work area

\$RETNWK macro [324](#)

route code

converting destination to [100](#)

routine

set environment [204](#)

## S

SAF (security authorization facility) invocation

\$SEAS macro [356](#)

SAF (security authorization facility) SAF

invoking [356](#)

save area

freeing (\$RETURN macro) [322](#)

obtaining (\$SAVE macro) [327](#)

returning (\$RETSAVE macro) [322](#)

storing registers (\$STORE macro) [377](#)

scan

\$SCAN facility [447](#)

\$SCANTAB [341](#)

backup storage [334](#)

initialization statement

\$SCAN [329](#)

input structure [447](#)

recursively [447](#)

scan [447](#)

tables (\$SCANTAB) [341](#)

scan of parameter statements [447](#)

schedule a checkpoint

\$CKPT macro [81](#)

search JOT for JOE

\$#GET macro [29](#)

secondary ASCB

retrieving, \$GETASCB macro [187](#)

sending to IBM

reader comments [xxv](#)

separator page

creating, \$PRPUT macro [296](#)

services provided by a macro

called routine [1](#)

inline code [1](#)

set address mode

\$AMODE macro [49](#)

set assembly environment

\$ENVIRON macro [156](#)

set busy system indicator

\$#BUSY macro [24](#)

\$QBUSY macro [300](#)

set IDAW

\$SETIDAW macro [364](#)

set interval timer

\$STIMER macro [371](#)

set recovery processing option

\$SETRP macro [365](#)

set work selection value

\$WSSETUP macro [402](#)

shared queue

synchronizing (\$QSUSE macro) [314](#)

shortcut keys [497](#)

- SJB
  - SJB (\$\$SJBFIND macro) [365](#)
  - SJB (\$\$SJBREQ macro) [368](#)
- specify processor as active
  - \$ACTIVE macro [46](#)
- specify processor inactive
  - \$DORMANT macro [141](#)
- SSI
  - end (\$\$SSIEND macro) [370](#)
- storage
  - acquiring (\$\$GETMAIN macro) [199](#)
  - cell acquiring (\$\$GETCEL macro) [191](#)
  - dumping (\$\$SDUMP macro) [355](#)
  - freeing (\$\$FREMAIN macro) [177](#)
  - freeing CSA [171](#)
- store register
  - \$STORE macro [377](#)
- subroutine
  - calling, \$CALL macro [64](#)
- subsystem interface
  - begin (\$\$SIBEGN macro) [369](#)
  - end (\$\$SSIEND macro) [370](#)
- subtask
  - initiate queueing [378](#)
- summary of changes xxvii, xxviii
- SUPPRES= macro parameter
  - \$DESTID macro [103](#)
- symptom record
  - creating [380](#), [381](#)
- synchronize
  - shared queues (\$\$QSUSE macro) [314](#)
- synchronize nit settings [273](#)
- synchronize use of a shared queue
  - \$QSUSE macro [314](#)
- syntax diagrams
  - how to read [2](#)
- SYSARM=
  - assembler instruction [248](#)
  - options [248](#)
- SYSARM= options
  - relation to \$MODULE [248](#)

## T

- table
  - macro selection [8](#)
- table generate
  - \$REPLYV macro [318](#)
- table map
  - \$SCANTAB [341](#)
  - \$TIDTAB [385](#)
  - PCE, \$PCETAB macro [281](#)
- table maps
  - \$DCTTAB macro [93](#)
- table pairs
  - description [425](#)
- tables
  - creating
    - \$TIDTAB entry [445](#)
    - \$TRACE macro [445](#)
- task
  - posting
    - \$XMPOST macro [421](#)
- termination of PCE

- termination of PCE (*continued*)
  - \$PCETERM macro [287](#)
- test interval timer
  - \$TTIMER macro [392](#)
- timer
  - setting (\$\$STIMER macro) [371](#)
  - testing (\$\$TTIMER macro) [392](#)
- trace
  - id table map (\$\$TIDTAB macro) [385](#)
- trace a JES2 activity
  - \$TRACE macro [388](#)
- track
  - address, acquiring (\$\$TRACK macro) [391](#)
- track group blocks
  - queueing, \$BLDTGB [63](#)
- trademarks [504](#)
- transmitting an NJE data area
  - across the network
    - \$NHDXMT macro [272](#)

## U

- UCB [181](#)
- UCB (unit control block)
  - freeing UCB parameter list [181](#)
  - obtaining its address (\$\$GETUCBS) [207](#)
- UCB address
  - obtaining, \$\$GETUCBS macro [207](#)
- unfinished JOE [41](#)
- unit record
  - allocation, \$ALLOC macro [47](#)
- user communication table (UCT) [428](#)
- user interface
  - ISPF [497](#)
  - TSO/E [497](#)
- USING on a register
  - establish [326](#)

## V

- variable field length operation
  - \$VFL macro [398](#)
- virtual page service
  - \$PGSRVC macro [290](#)

## W

- wait for a JES2 event
  - \$WAIT macro [399](#)
- wake up the execution PCE
  - \$POSTXEQ macro [295](#)
- work area
  - obtaining (\$\$GETWORK macro) [209](#)
- work JOEs
  - formatting [23](#)
  - remove a pair from the JOT [42](#)
  - replace a characteristics JOE [44](#)
  - replace a work JOE [44](#)
- work/characteristic JOE pair [20](#)
- write to operator
  - \$SWTO JES2 subtask [18](#)
  - \$SWTO macro [18](#)
  - \$SWTOR macro with reply [20](#)

write to operator (*continued*)  
command processor  
    \$CWTO macro [89](#)  
JES2 subtask with reply  
    \$\$WTOR macro [20](#)  
message area  
    \$MSG macro [262](#)







Product Number: 5650-ZOS

SA32-0996-50

