USING REXX COMMAND FILES WITH TEAMCONNECTION MVS BUILD SCRIPTS FOR PL/I PROGRAMS

Document Number TR 29.2235

Mike Petersen
Jeri Petersen

TeamConnection Integration
IBM Software Solutions
Research Triangle Park, N.C.

(c) Copyright International Business Machines Corporation 1996.
All rights reserved.

TRADEMARKS AND SERVICE MARKS

The following terms are trademarks or service marks of the IBM Corporation in the United States or other countries:

CICS

DB2

IBM

IMS

OS/2

TeamConnection

VisualAge

The following terms are trademarks or service marks of other companies:

Object Store	Object Design, Inc.

© Copyright IBM Corp. 1997

ABSTRACT

This report introduces the concept of using REXX command files for use with TeamConnection MVS build scripts. Using REXX command files for TeamConnection MVS builds allows you to use the power of REXX to provide function that is not possible in the normal JCL-like build scripts for MVS. This includes:

Only returning messages to TeamConnection when there is a problem.

Concatenating a program product include library that is maintained on the MVS host with other include members that are maintained within TeamConnection.

Changing compiler or linkage editor options based on the work area, driver, or release name.

Changing the DB2 system id based on a work area, driver, or release name.

Trapping messages from DB/2 where multiple messages might be returned.

Handling multiple outputs (for example, both a DBRM and object code).

Needing to get the link edit and bind to occur at the same time so that the date/time stamp for the load module and package are not out of date.

The specific examples are for PL/I, DB2, and CICS. However, the techniques can be applied to other languages such as COBOL and other environments such as IMS or TSO. Refer to *TR29.2196 - Using REXX Command Files with TeamConnection MVS Build Scripts* for information on handling COBOL.

ACKNOWLEDGMENTS

The authors would also like to acknowledge the following people who worked on build scripts for PL/I before this Technical Report was started:

Tom Loeber
Mehdi Sanayei

ITIRC KEYWORDS

Build agent

Build processor

Build server

CICS

PL/I

© Copyright IBM Corp. 1997

Configuration Management

DB2

Drivers

Library Management

REXX

TeamConnection

Work areas

ABOUT THE AUTHORS

Mike Petersen is an IBM software developer for TeamConnection integration in Raleigh, North Carolina. Mike was an SCLM software developer for six years. He has worked in systems, application, and other product programming areas. He also has had assignments with world trade, IBM faculty loan program, technical education, and technical support for marketing.

Mike received a BS in mathematics with a physics minor in 1972 and an MS in computer science in 1974 from Purdue University. In 1979 he received an MBA from Marist College.

Jeri Petersen is an IBM software developer for TeamConnection integration in Raleigh, North Carolina. Jeri was in VisualAge Generator development for nine years. She has also worked in application development for IBM, as well as for other companies.

Jeri received a BS in mathematics with a physics minor in 1972 from Purdue University, an MS in computer science in 1974 from Rutgers University, and an MBA in 1979 from Marist College.

© Copyright IBM Corp. 1997

CONTENTS

TRADEMARKS AND SERVICE MARKS iii

ABSTRACT v
Acknowledgments v
ITIRC Keywords v

ABOUT THE AUTHORS vii

Figures xi

BEFORE WE BEGIN 1
Disclaimer - Notice to Users of the Source Code Examples 2

OVERVIEW 3
RUNPGM 3
RUNPGMT 6

HOW THE BUILDER DEFINITION, IRXJCL, AND THE REXX COMMAND FILE WORK 19
Defining the Builder 19
Notes on mvspli.jcl (the Build Script) 19
Notes on mvsrxpli.rxp (the REXX Command File) 20

SAMPLE PARTS AND BUILD TREES FOR THE BASIC TECHNIQUE 23
Simple PL/I Program Include Member 23
PL/I Program, Link Edit Statements, and Build Tree 24
PL/I with DB2 Program, Link Edit Statements, and Build Tree 26
PL/I with CICS Program, Link Edit Statements, and Build Tree 29
PL/I with DB2 and CICS Program, Link Edit Statements, and Build Tree 31
Sibling Output Build Trees 33

PUTTING IT ALL TOGETHER 37
Create a New Family 37
Start the TeamConnection Client 40
Create Components 41
Create a Release 42
Create Builders 42
Create A Parser 45
Open a Feature and Create a Work Area 46

© Copyright IBM Corp. 1997

Create Parts 47
Create Build Tree 50
Start the MVS Build Agent and Build Processor 53
Build the Object Modules 54
Build the Load Modules 55

VARIATIONS IN mvspli.jcl AND mvsrxpli.rxp 57
Keeping mvspli.jcl at the MVS Host 57
Keeping mvsrxpli.rxp at the MVS Host 58
Supporting Program Product Include Libraries 58
Supporting Program Product Include Libraries Concatenated with TeamConnection Copybooks 59
Setting PL/I Compiler Options - Hardcoding 60
Setting PL/I Compiler Options - From Builder Definition 61
Setting PL/I Compiler Options - Based on Work Area Name 62
Reducing the Number of Build Messages 64

HINTS AND TIPS FOR MVS BUILDS 65

Appendix A. BUILDER, BUILD SCRIPT, AND REXX FOR LINK EDIT AND BIND . . . 69
Defining the Builder 69
Notes on mvslnk.jcl (the Build Script) 70
Notes on mvsrplnk.rxp (the REXX Command File) 73
Variations in Link and Bind 85
Combining mvsrxpli.rxp with mvsrplnk.rxp 85
Using a BIND Command File 86
Reducing the Number of Build Messages 89

Appendix B. SAMPLE REXX TO CREATE BUILD TREES 91
Variations in Creating the Build Trees 94

Appendix C. GLOSSARY 95

FIGURES

1. Sample JCL Using LE/370 for RUNPGM . . . 3
2. Sample JCL to Compile a PL/I Program . . . 4
3. Sample JCL-Like Build Script to Compile a PL/I Program . . . 5
4. Sample JCL Using LE/370 for RUNPGMT . . . 8
5. mvspli Builder Definition When Using REXX Command File for a PL/I Compile . . . 8
6. mvspli.jcl - Specialized Build Script Using IRXJCL to Pass Control to REXX Command File . . . 9
7. Build Tree for a PL/I Compile Using REXX Command File . . . 10
8. mvsrxpli.rxp - REXX Command File Supporting DB2, CICS, and PL/I Programs . . . 11
9. PLIPGM.INC - PL/I Include Member . . . 23
10. PLIPGM.PLI - PL/I Program . . . 24
11. PLIPGM.LNK - PL/I Link Edit Statements . . . 24
12. PLIPGM - Build Tree for PL/I Program . . . 25
13. PLIPGM - Connect Commands to Create Build Tree . . . 25
14. PLIDB2 - PL/I Program with DB2 . . . 26
15. PLIDB2.LNK - PL/I with DB2 Link Edit Statements . . . 26
16. PLIDB2 - Build Tree for PL/I Program with DB2 . . . 27
17. PLIDB2 - Connect Commands to Create Build Tree . . . 28
18. PLICICS - PL/I Program with CICS . . . 29
19. PLICICS.LNK - PL/I with CICS Link Edit Statements . . . 29
20. PLICICS - Build Tree for PL/I Program with CICS . . . 30
21. PLICICS - Connect Commands to Create Build Tree . . . 30
22. PLIDCIC - PL/I Program with DB2 and CICS . . . 31
23. PLIDCIC.LNK - PL/I with DB2 and CICS Link Edit Statements . . . 31
24. PLIDCIC - Build Tree for PL/I Program with DB2 and CICS . . . 32
25. PLIDCIC - Connect Commands to Create Build Tree . . . 33
26. Build Trees for Sibling Outputs PLIDB2.OBJ and PLIDB2.DBR . . . 34
27. Builder Information . . . 43
28. mvslnk.jcl - Build Script for Link Edit and Bind . . . 70
29. mvsrplnk.rxp - REXX Command File Supporting Link Edit and Bind . . . 74
30. REXX Command File to Create Build Trees . . . 92

© Copyright IBM Corp. 1997

BEFORE WE BEGIN

This report is written for the TeamConnection user who needs to create build scripts for MVS. Experience should include some familiarity with TeamConnection concepts, MVS build processors, and the REXX language.

The sample JCL shown for RUNPGM and RUNPGMT has been customized to use LE/370.

The names of the programs and builders have been chosen to indicate the type of processing they perform. Any names could be used in their place.

The family server was on an OS/2 machine.

The technique described in this Technical Report was tested with TeamConnection Version 2.0.

© Copyright IBM Corp. 1997

DISCLAIMER - NOTICE TO USERS OF THE SOURCE CODE EXAMPLES

THIS TECHNICAL REPORT PROVIDES THE SOURCE CODE EXAMPLES, BOTH INDIVIDUALLY AND AS ONE OR MORE GROUPS, "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE SOURCE CODE EXAMPLES, BOTH INDIVIDUALLY AND AS ONE OR MORE GROUPS, IS WITH YOU. SHOULD ANY PART OF THE SOURCE CODE EXAMPLES PROVE DEFECTIVE, YOU (AND NOT IBM OR AN AUTHORIZED DEALER) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

OVERVIEW

TeamConnection supplies 2 MVS jobs that can be used to run an MVS build processor:

RUNPGM, described in “RUNPGM.”
RUNPGMT, described in “RUNPGMT” on page 6.

RUNPGM

RUNPGM, shown in Figure 1, runs the TeamConnection MVS build processor, TEAMPROC, as an MVS batch job. This technique supports build scripts written entirely in a JCL-like language.

```
//your job card
//                                        /
//  PROGRAM:    RUNPGM                                                  /
//  JCL to start TeamConnection MVS Build Server                  /
//                                        /
//    Some dataset names might need to be modified                /
//    according to your system's customization                   /
//                                        /
//RUNPGM     EXEC  PGM=TEAMPROC,
// PARM='ENVAR("_CEE_ENVFILE=DD:EDCENV")/-S @329 1 -U VIO -K IBM-1 47',
// REGION=4M
//STEPLIB DD DSN=userid.TEAMC.LOADLIB,DISP=SHR
//             DD  DSN=DSNA.DSNLOAD,DISP=SHR
//             DD  DSN=SYS1.CEE.SCEERUN,DISP=SHR
//             DD  DSN=SYS1.PLI.PLICOMP,DISP=SHR
//  userid.TEST.JCL must be allocated as a PDS with lrecl=8 ,
//  and recfm=FB
//TEAMPROC DD DSN=userid.TEAMC.TEST.JCL,DISP=SHR
//EDCENV     DD  DSN=userid.TEAMC.ENVVAR,DISP=SHR
//STDOUT     DD  SYSOUT=
//STDERR     DD  SYSOUT=
//SYSUDUMP DD SYSOUT=
//CEEDUMP   DD  SYSOUT=
//
```

Figure 1. Sample JCL Using LE/370 for RUNPGM

Build scripts written in the JCL-like language are easy to create from existing JCL. For example, Figure 2 on page 4, shows the JCL used to do a PL/I compile without

© Copyright IBM Corp. 1997

TeamConnection involved. Figure 3 on page 5 shows the corresponding build script for TeamConnection. TCPARM and TCEXT are special TeamConnection keywords that indicate parameters to be passed and file extensions for parts respectively.

```
//your job card
// ------------------------------------------------------------------ /
//  PROGRAM:    mvspli.jcl
//  PL/I for MVS - Compile Only
//
// ------------------------------------------------------------------ /
//PLICOMP   EXEC  PGM=IEL AA,PARM='INCLUDE,OBJECT,NODECK'
//STEPLIB   DD  DISP=SHR,DSN=DSNA.DSNLOAD
//              DD  DISP=SHR,DSN=SYS1.PLI.PLICOMP
//SYSIN      DD  DISP=SHR,DSN=userid.PRIVATE.SOURCE(PLIAPP)
//SYSLIB     DD  DISP=SHR,DSN=userid.PRIVATE.SOURCE
//SYSPRINT DD DISP=SHR,DSN=userid.PRIVATE.SYSPRINT
//SYSLIN     DD  DISP=OLD,DSN=userid.TCMVS.OBJ(PLIAPP)
/
//SYSUT1     DD    DSN=&&SYSUT1,UNIT=SYSDA,
//                    SPACE=(1 24,(2  ,5 ),,CONTIG,ROUND),DCB=BLKSIZE=1 24
```

Figure 2. Sample JCL to Compile a PL/I Program

```
// ------------------------------------------------------------------ /
//  PROGRAM:    mvspli.jcl
//  PL/I for MVS - Compile Only
//
// ------------------------------------------------------------------ /
//PLICOMP   EXEC  PGM=IEL AA,
// PARM='&TCPARM'
//SYSIN      DD    TCEXT=PLI,DISP=(NEW,DELETE),
//    SPACE=(32   ,(3 ,1 )),UNIT=SYSDA,
//    DCB=(RECFM=FB,LRECL=8 ,BLKSIZE=32  )
//SYSLIB     DD    TCEXT=INC,DISP=(NEW,DELETE),
//    SPACE=(32   ,(3 ,1 ,5)),UNIT=SYSDA,
//    DCB=(RECFM=FB,LRECL=8 ,BLKSIZE=32  )
//SYSPRINT DD    TCEXT=TCOUT,DISP=(NEW,DELETE),
//    SPACE=(133  ,(3 ,3 )),UNIT=SYSDA,
//    DCB=(RECFM=FBA,LRECL=133,BLKSIZE=133 )
//SYSLIN     DD    TCEXT=OBJ,UNIT=SYSDA,
//    DISP=(NEW,DELETE),SPACE=(32   ,(3 ,1 )),
//    DCB=(RECFM=FB,LRECL=8 ,BLKSIZE=32  )
//
//SYSUT1     DD    DSN=&&SYSUT1,UNIT=SYSDA,
//                    SPACE=(1 24,(2  ,5 ),,CONTIG,ROUND),DCB=BLKSIZE=1 24
```

Figure 3. Sample JCL-Like Build Script to Compile a PL/I Program

RUNPGMT

RUNPGMT, shown in Figure 4 on page 8, runs the TeamConnection MVS build processor, TEAMPROC, under an MVS/TSO session from within an MVS batch job. RUNPGMT was originally developed for use with VisualAge Generator. However, as explained in the sections that follow, RUNPGMT can be used for other languages. RUNPGMT is used to accomplish the build in the following way.

The RUNPGMT job, shown in Figure 4 on page 8, runs the MVS build processor and:

1. Starts IKJEFT01 (an MVS/TSO session).
2. Calls TEAMPROC from within the MVS/TSO session.
3. When the build starts from TeamConnection, the build script specified for the builder is placed in the data set identified by the TEAMPROC DD statement.

The builder definition, shown in Figure 5 on page 8, identifies the following:

- The source file for the build script.
- The parameters for the build script. When using a REXX command file for MVS builds, the first parameter specifies the name of the REXX command file that is to run during the build. The other parameters are passed to the REXX command file.

The builder definition in this example is for a PL/I compile.

The build script, shown in Figure 6 on page 9, causes the following to happen:

- Gives control to program IRXJCL. IRXJCL allows a REXX command file to run under the MVS/TSO session.
- The REXX command file must be in the data set identified by the SYSEXEC DD statement.
- Other DD statements in the build script support inputs or outputs for the build that must be transferred from / to TeamConnection. In Figure 6 on page 9, the DD statements provide the inputs and outputs needed for combinations of a DB2 precompile, a CICS translate, and a PL/I compile.

This same build script can be used to support several builder definitions. See “Notes on mvspli.jcl (the Build Script)” on page 19 for more a detailed explanation of this build script.

The build tree, shown in Figure 7 on page 10, includes an input part with a file extension of .rxp. Using TCEXT=RXP on the SYSEXEC DD statement of the build script causes the .rxp part from the build tree to be placed in the temporary data set associated with the SYSEXEC DD statement. The build tree in this example is for a PL/I compile.

The mvsrxpli.rxp part shown in Figure 8 on page 11, does the actual function needed for the build. The mvsrxpli.rxp part handles combinations of a DB2 precompile, a CICS translate, and a PL/I compile.

This same REXX command file can be used to support several builder definitions just by varying the parameters specified in the builder definition. See “Notes on mvsrxpli.rxp (the REXX Command File)” on page 20 for more a detailed explanation of this REXX command file.

Using build scripts for IRXJCL that invokes a REXX command file has the following advantages:

It is easier to combine several steps into a single build script. For example, the REXX command file can pass the output of the DB2 precompiler to the CICS translator and from the CICS translator to the PL/I compiler.

Better performance if the intermediate outputs do not need to be saved in TeamConnection because they do not need to be passed back to the build agent.

Smaller build tree because the intermediate outputs do not need to be shown because they do not need to be saved in TeamConnection.

The same REXX command file can be used to handle several combinations of translators. For example, the REXX command file shown in Figure 8 on page 11 handles the following:

- PL/I
- CICS and PL/I
- DB2 and PL/I
- DB2, CICS, and PL/I

```
//your job card
//                                        /
//  PROGRAM:    RUNPGMT                                                  /
//  JCL to start TeamConnection MVS Build Server under          /
//  a TSO environment.    Primary use is intended for               /
//  VisualGen support.                                                    /
//                                        /
//     Some dataset names might need to be modified                 /
//     according to your system's customization                     /
//                                        /

//RUNPGMT EXEC PGM=IKJEFT 1,REGION=6M,DYNAMNBR=3
//STEPLIB DD DSN=DSNA.DSNLOAD,DISP=SHR
//             DD  DSN=userid.TEAMC.LOADLIB,DISP=SHR
//             DD  DSN=SYS1.CEE.SCEERUN,DISP=SHR
//             DD  DSN=SYS1.COB2COMP,DISP=SHR
//  userid.TEST.JCL must be allocated as a PDS with lrecl=8 ,
//  and recfm=FB
//TEAMPROC DD DSN=userid.TEAMC.TEST.JCL,DISP=SHR
//EDCENV     DD    DSN=userid.TEAMC.ENVVAR,DISP=SHR
//STDOUT     DD  SYSOUT=
//STDERR     DD  SYSOUT=
//SYSUDUMP DD SYSOUT=
//CEEDUMP   DD  SYSOUT=
//SYSTSPRT DD SYSOUT=
//SYSTSIN DD
 CALL 'userid.TEAMC.LOADLIB(TEAMPROC)'    +
       'ENVAR("_CEE_ENVFILE=DD:EDCENV")/-S @32911 -U SYSDA -K IBM-1 47'
//
```

Figure 4. Sample JCL Using LE/370 for RUNPGMT

```
Builder          mvspli
Release          release1
Script           mvspli.jcl
Environment      mvstso
Comp. oper.      <=
RC value         4
File type        text
Source file      e:\tcdemo2\plirexx\scripts\mvspli.jcl
Parameters       'MVSRXPLI N N Y'
Timeout          1
```

Figure 5. mvspli Builder Definition When Using REXX Command File for a PL/I Compile

```
//REXX        EXEC    PGM=IRXJCL,
// PARM=&TCPARM
//
//  BUILD SCRIPT FOR EXECUTING A REXX PROGRAM TO DO PL/I COMPILES
//
//  TEAMPROC MUST RUN IN A TSO ADDRESS SPACE
//
//  IRXJCL RUNS THE PROGRAM NAMED IN THE FIRST PARAMETER OF &TCPARM
//  AS A REXX PROGRAM AND PASSES THE REMAINING PARAMETERS
//
//  IRXJCL LOOKS INTO SYSEXEC FOR THE REXX PROGRAM
//
//SYSEXEC   DD  TCEXT=RXP,DISP=(NEW,DELETE),                          (1)
//             UNIT=VIO,SPACE=(32   ,(3 ,1 ,5)),
//             DCB=(RECFM=FB,LRECL=8 ,BLKSIZE=32  )
//DBRM        DD  TCEXT=DBR,DISP=(NEW,DELETE),
//             UNIT=SYSDA,SPACE=(32   ,(3 ,1 ,5)),
//             DCB=(RECFM=FB,LRECL=8 ,BLKSIZE=616 )
//SOURCE     DD  TCEXT=PLI,DISP=(NEW,DELETE),
//             UNIT=SYSDA,SPACE=(32   ,(3 ,1 )),
//             DCB=(RECFM=FB,LRECL=8 ,BLKSIZE=32  )
//INCSRC     DD  TCEXT=INC,DISP=(NEW,DELETE),                         (2)
//             UNIT=SYSDA,SPACE=(32   ,(3 ,1 ,5)),
//             DCB=(RECFM=FB,LRECL=8 ,BLKSIZE=32  )
//OBJECT     DD  TCEXT=OBJ,DISP=(NEW,DELETE),
//             UNIT=SYSDA,SPACE=(32   ,(3 ,1 )),
//             DCB=(RECFM=FB,LRECL=8 ,BLKSIZE=32  )
//ERRLIST   TCEXT=TCOUT,DISP=(NEW,DELETE),
//             UNIT=SYSDA,SPACE=(133  ,(3 ,3 )),
//             DCB=(RECFM=FB,LRECL=133,BLKSIZE=133 )
//OUTLIST   DD                                                        (3)
&TCOUTPUT
//
```

Figure 6. mvspli.jcl - Specialized Build Script Using IRXJCL to Pass Control to REXX Command File

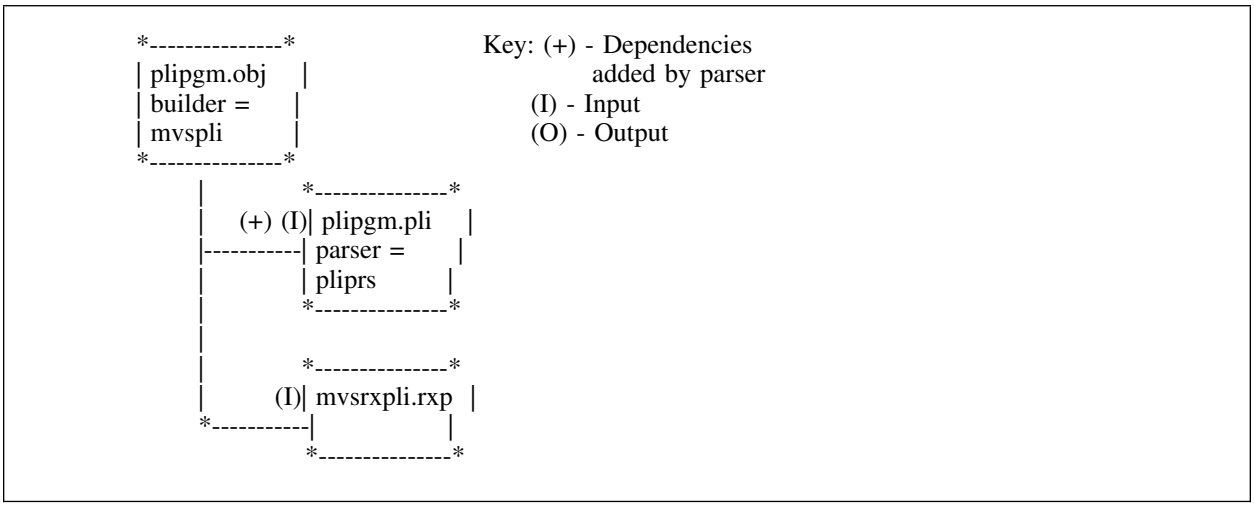

Figure 7. Build Tree for a PL/I Compile Using REXX Command File

```
/  REXX  /
TRACE INTERMEDIATES

parse arg db2 cics plicmp otherparms                                    (1)
/                                                  /
/   INITIALIZATION                                                                  /
/                                                  /
                                                                                  (2)
wspc            = 5                                  /  Space allocation default      /

dsnload         = 'DSNA.DSNLOAD'                     /  DB2 libraries                    /

dfhload         = 'CICS33 .SDFHLOAD'                 /  CICS libraries / variables  /
cics_suffix = '1$'

plicomp         = 'SYS1.PLI.PLICOMP'                 /  PLI libraries                    /

/                                                  /
/  RETRIEVE DATASET NAMES FOR FILES DEFINED IN THE BUILD SCRIPT                /
/                                                  /
                                                                                  (3)
dsinfo = getdsn_last('DBRM')
if dsinfo >   then
   call errorhndlr GETDBRM,dsinfo
dbrmlib = SYSDSNAME

dsinfo = getdsn_last('SOURCE')
if dsinfo >   then
   call errorhndlr GETSOURCE,dsinfo
srclib = SYSDSNAME

dsinfo = getdsn_last('INCSRC')
if dsinfo >   then
   call errorhndlr GETINCSRC,dsinfo
cpysrc = SYSDSNAME

dsinfo = getdsn_last('OBJECT')
if dsinfo >   then
   call errorhndlr GETOBJECT,dsinfo
objlib = SYSDSNAME
```

Figure 8 (Part 1 of 7). mvsrxpli.rxp - REXX Command File Supporting DB2, CICS, and PL/I Programs

```
/                                                  /
/  FORCE DB2, CICS, AND PLICMP PARAMETERS TO Y OR N                          /
/                                                  /
                                                                    (4)
if db2          <> 'Y' then
   db2              =  'N'
if cics         <> 'Y' then
   cics             =  'N'
if plicmp <> 'Y' then
   plicmp         =  'N'

/                                                  /
/  CALL APPROPRIATE ROUTINES                                                 /
/                                                  /
                                                                    (5)
if db2 = 'Y' then
   do
      call  getdbrmnam
      call  precompile
   end
if cics = 'Y' then
   call  translate
if plicmp = 'Y' then
   call  compile

exit
```

Figure 8 (Part 2 of 7). mvsrxpli.rxp - REXX Command File Supporting DB2, CICS, and PL/I Programs

```
precompile:
/                                          /
/    ALLOCATE THE DATASETS NECESSARY FOR THE DB2 PRECOMPILE.                 /
/    THE DATASET ALLOCATED TO THE SYSCIN DD IS PASSED TO EITHER THE      /
/    THE TRANSLATE OR COMPILE SUBROUTINE DEPENDING ON WHETHER CICS IS  /
/    INVOLVED OR NOT.                                                        /
/                                          /
                                                                    (6)
"ALLOC FI(DBRMLIB)    DA('"dbrmlib"("dbrmmbr")') OLD REUSE"
"ALLOC FI(SYSCIN)      BLOCK(8  ) SPACE("wspc","wspc") UNIT(SYSDA) REUSE"
"ALLOC FI(SYSLIB)      DA('"cpysrc"') SHR REUSE"                    (7)
"ALLOC FI(SYSPRINT) DUMMY REUSE"
"ALLOC FI(SYSTERM)    BLOCK(8  ) SPACE("wspc","wspc") UNIT(SYSDA) REUSE"
"ALLOC FI(SYSUT1)      BLOCK(8  ) SPACE("wspc","wspc") UNIT(VIO) REUSE"
"ALLOC FI(SYSUT2)      BLOCK(8  ) SPACE("wspc","wspc") UNIT(VIO) REUSE"
"ALLOC FI(SYSIN)       DA('"srclib"') SHR REUSE"

"CALL '"dsnload"(DSNHPC)' 'HOST(PLI)'"                                (8)
precompile_rc = rc

"FREE FI(DBRMLIB,SYSLIB,SYSPRINT,SYSUT1,SYSUT2,SYSIN)"
if precompile_rc <  | precompile_rc > 4 then                    (9)
  do
     "FREE  FI(SYSCIN)"
      call errorhndlr PRECOMPILE,precompile_rc,SYSTERM
  end
else
   call TCOUThndlr PRECOMPILE,precompile_rc,SYSTERM                (1 )

return
```

Figure 8 (Part 3 of 7). mvsrxpli.rxp - REXX Command File Supporting DB2, CICS, and PL/I Programs

```
translate:
/                                          /
/    ALLOCATE THE DATASETS NECESSARY FOR THE CICS TRANSLATOR.              /
/                                          /
                                                                    (11)
"ALLOC FI(SYSPRINT)    SPACE("wspc","wspc") UNIT(SYSDA) REUSE"
"ALLOC FI(SYSPUNCH)    BLOCK(8  ) SPACE("wspc","wspc") UNIT(SYSDA) REUSE"

/                                /
/  IF DB2 WAS USED                                  /
/  THEN                                             /
/      ALLOCATE SYSCIN TO SYSIN                        /
/  ELSE                                            /
/      ALLOCATE SYSIN TO THE BUILD OBJECT           /
/                                /
                                                                    (12)
if db2 = 'Y' then
   call realloc SYSCIN,SYSIN
else
   "ALLOC FI(SYSIN)      DA('"srclib"') SHR REUSE"

"CALL '"dfhload"(DFHEPP"cics_suffix")'",                          (13)
        '"NOSOURCE"'
translate_rc = rc

"FREE FI(SYSCIN SYSIN)"
if translate_rc <  | translate_rc > 4 then                    (14)
  do
     "FREE  FI(SYSPUNCH)"
      call errorhndlr TRANSLATE,translate_rc,SYSPRINT
  end
else
   call TCOUThndlr TRANSLATE,translate_rc,SYSPRINT                (15)

return
```

Figure 8 (Part 4 of 7). mvsrxpli.rxp - REXX Command File Supporting DB2, CICS, and PL/I Programs

```
compile:
/                                                  /
/    COMPILE ALLOCATES THE DATASETS NECESSARY TO COMPILE THE                    /
/   REQUESTED  OBJECT.                                                          /
/                                                  /
/                                  /
/  IF CICS WAS USED                                  /
/  THEN                                              /
/      ALLOCATE SYSPUNCH TO SYSIN                        /
/  ELSE                                              /
/      IF DB2 WAS USED                                 /
/     THEN                                           /
/         ALLOCATE SYSCIN TO SYSIN                      /
/     ELSE                                           /
/         ALLOCATE SYSIN TO THE BUILD OBJECT        /
/                                  /
                                                                                (16)
if cics = 'Y' then
   call realloc SYSPUNCH,SYSIN
else
   if db2 = 'Y' then
      call realloc SYSCIN,SYSIN
  else
      "ALLOC FI(SYSIN) DA('"srclib"') SHR REUSE"
                                                                                (17)
"ALLOC FI(SYSLIB)     DA('"cpysrc"') SHR REUSE"                                 (18)
"ALLOC FI(SYSLIN)     DA('"objlib"') SHR REUSE"
"ALLOC FI(SYSUT1)     BLOCK(8  ) SPACE("wspc","wspc") UNIT(VIO) REUSE"
"ALLOC FI(SYSUT2)     BLOCK(8  ) SPACE("wspc","wspc") UNIT(VIO) REUSE"
"ALLOC FI(SYSUT3)     BLOCK(8  ) SPACE("wspc","wspc") UNIT(VIO) REUSE"
"ALLOC FI(SYSUT4)     BLOCK(8  ) SPACE("wspc","wspc") UNIT(VIO) REUSE"
"ALLOC FI(SYSPRINT) BLOCK(8  ) SPACE("wspc","wspc") UNIT(SYSDA) REUSE"
                                                                                (19)
pliopts = 'INCLUDE,MACRO,OBJECT,NODECK,MAP,OPT(2),NOSOURCE'
if cics = 'Y' then
   pliopts = pliopts || ',SYSTEM(CICS)'

"CALL '"plicomp"(IEL AA)' '"pliopts"'"

compile_rc = rc

"FREE FI(SYSUT1,SYSUT2,SYSUT3,SYSUT4)"
"FREE FI(SYSPUNCH SYSCIN SYSIN,SYSLIB,SYSLIN)"
if compile_rc <   | compile_rc > 4 then                                  (2 )
   call errorhndlr COMPILE,compile_rc,SYSPRINT
else
   call TCOUThndlr COMPILE,compile_rc,SYSPRINT                                  (21)

return
```

Figure 8 (Part 5 of 7). mvsrxpli.rxp - REXX Command File Supporting DB2, CICS, and PL/I Programs

```
realloc: procedure
/                                                  /
/    REALLOCATE A FILE TO A PREVIOUS DATA SET                                  /
/                                                  /
parse arg oldddname, newddname
dsinfo = getdsn_last(oldddname)
"ALLOC FI("newddname") DSN("'SYSDSNAME'") SHR REUSE"
return

getdsn_last: procedure expose sysdsname sysmsglvl1 sysmsglvl2
/                                                  /
/    DETERMINE DATA SET NAME OF A TEMPORARY FILE                               /
/                                                  /
parse arg ddname parm_rest
getdsn ddname
getdsn_rc = rc
if getdsn_rc <   | getdsn_rc >   then
   sysdsname = ''
else
  do
     last_dsn = sysdsname.
     sysdsname = sysdsname.last_dsn
  end
return getdsn_rc


getdbrmnam: procedure expose dbrmmbr                                          (22)
/                                                  /
/    DETERMINE NAME OF PL/I SOURCE FOR USE AS DBRM MEMBER NAME                 /
/                                                  /
dsinfo = getdsn_last('OUTLIST')
if dsinfo >   then
   call errorhndlr GETOUTLIST,dsinfo

myoutput. = ''
"ALLOC FI(OUTLISTX) DA("'SYSDSNAME'") SHR REUSE"
"execio   diskr OUTLISTX (stem myoutput. open finis)"

/  OUTLIST contains one line for each output in the build tree.            /
/  Because the file extension is removed, the OBJ and DBRM outputs       /
/  cannot be distinguished.    Use the naming convention of xxxxx.OBJ     /
/  and xxxxx.DBR for the object module and DBRM respectively.               /
/  This means it does not matter which line of OUTLIST is used to        /
/  set dbrmmbr.                                                                /

dbrmmbr = word(myoutput.1,1)

return
```

Figure 8 (Part 6 of 7). mvsrxpli.rxp - REXX Command File Supporting DB2, CICS, and

```
TCOUThndlr: procedure expose sysmsglvl1 sysmsglvl2                    (23)
/                                                 /
/    SAVE PRINT LISTINGS FROM EACH STEP TO RETURN TO TEAMCONNECTION      /
/                                                 /
parse arg function, return_code, copyddname
dsinfo = getdsn_last('ERRLIST')
"ALLOC FI(ERROUT) DA('"SYSDSNAME"') MOD REUSE"
if copyddname <> '' then
  do
      "execio   diskr "copyddname" (stem errvar. open finis)"
      "execio   diskw errout (STEM errvar. open finis)"
     "FREE  FI("copyddname")"
  end
else
   'EXECIO 2 DISKW ERROUT (STEM sysmsglvl OPEN FINIS)'
"FREE FI(ERROUT)"

return


errorhndlr: procedure expose sysmsglvl1 sysmsglvl2                    (24)
/                                                 /
/    ISSUE ERROR MESSAGE FOR FUNCTION, ERROR CODE, AND DATA SET          /
/                                                 /
parse arg function, return_code, copyddname
dsinfo = getdsn_last('ERRLIST')
"ALLOC FI(ERROUT) DA('"SYSDSNAME"') MOD REUSE"
if copyddname <> '' then
  do
      "execio   diskr "copyddname" (stem errvar. open finis)"
      "execio   diskw errout (STEM errvar. open finis)"
     "FREE  FI("copyddname")"
  end
else
   'EXECIO 2 DISKW ERROUT (STEM sysmsglvl OPEN FINIS)'
"FREE FI(ERROUT)"
exit(return_code)
```

Figure 8 (Part 7 of 7). mvsrxpli.rxp - REXX Command File Supporting DB2, CICS, and PL/I Programs

HOW THE BUILDER DEFINITION, IRXJCL, AND THE REXX COMMAND FILE WORK

This section describes how the builder definition, the build script, and the REXX command file work together to control the build for the object module.

DEFINING THE BUILDER

When IRXJCL is used in the build script, the parameters for the builder define which REXX command file is used during the build. For example, in the builder definition:

```
Builder          mvspli
Release          release1
Script           mvspli.jcl
Environment      mvstso
Comp. oper.      <=
RC value          4
File type        text
Source file      e:\tcdemo2\plirexx\scripts\mvspli.jcl
 Parameters      'MVSRXPLI N N Y'
Timeout           1
```

the parameters are:

MVSRXPLI Name of the REXX command file that is run by IRXJCL. IRXJCL always expects the first parameter to be the name of the REXX command file.

N N Y Correspond to the parameters expected by MVSRXPLI.

The **Script** name is the name of the build script in Figure 6 on page 9.

NOTES ON MVSPLI.JCL (THE BUILD SCRIPT)

The following notes correspond to the numbers in the right margin of the mvspli.jcl build script, shown in Figure 6 on page 9. The notes explain concepts used in the build script.

Note:

1. The SYSEXEC DD statement must point to the library where the REXX command file resides. In this example, mvsrxpli.rxp is included in the build tree for the part. Because mvsrxpli.rxp has the file extension of .rxp, TCEXT=RXP causes mvsrxpli.rxp to be placed in the temporary data set allocated to the SYSEXEC DD statement. This makes mvsrxpli.rxp available to the IRXJCL program.

© Copyright IBM Corp. 1997

2. The INCSRC DD statement is the temporary library where include members that are maintained on TeamConnection will be placed. This assumes that all the include members have the file extension of .inc.
3. The OUTLIST DD statement is used by mvsrxpli.rxp to obtain the names of the output parts. &TCOUTPUT results in multiple lines of in-stream data, with each line corresponding to one output part. The output parts included are based on the build tree. The file extensions are not included.

   Using a naming convention where the object module name and DBRM member have the same part name except for the file extension allows mvsrxpli.rxp to determine the name of the DBRM member that is created by the precompiler. For example, if the object module name is plidb2.obj, the corresponding DBRM name is plidb2.dbr. Thus mvsrxpli.rxp does not need to determine which of the lines within the OUTPUT DD statement is really the DBRM.

NOTES ON MVSRXPLI.RXP (THE REXX COMMAND FILE)

The following notes correspond to the numbers in the right margin of the mvsrxpli.rxp REXX command file, shown in Figure 8 on page 11. The notes explain concepts used in the REXX command file.

Note:

1. The parameters for mvsrxpli.rxp are:

   db2 Y/N flag indicating whether a DB2 precompile is required

   cics Y/N flag indicating whether a CICS translate is required

   plicmp Y/N flag indicating whether a PL/I compile is required

   otherparms Other parameters could be added in the future. For example, to control the options for the DB2 precompiler, CICS translator, and/or PL/I compiler.
2. The initialization statements set the names of the MVS host libraries and other variables used in the REXX command file. You might need to change the names of the libraries to match the data set naming conventions for your organization.
3. Determine the names of the temporary data sets that were allocated to various DD statements.
4. Insure that the db2, cics, and plicmp parameters have only Y or N values.
5. Call the correct (internal) subroutines based on the parameters that were passed to mvsrxpli.rxp.
6. Allocate data sets that are used by the DB2 precompiler. The data sets allocated here correspond to the data sets that are used in your current JCL for the DB2 precompiler.

7. Allocate the temporary data set that contains the include members that are maintained on TeamConnection.
8. Invoke the DB2 precompiler, specifying the parameters to be used. The parameters are hard-coded in this example.
9. If there was an error, free the output data set where the PL/I source that was expanded by the DB2 precompiler is normally placed. Also call the errorhndlr routine to format the DB2 precompiler messages to go back to TeamConnection. The errorhndlr routine exits from the mvsrxpli.rxp REXX command file.
10. If there were no errors, call the TCOUThndlr routine to save the messages and other information that were written to the SYSTERM file by the DB2 precompiler. The TCOUThndlr routine saves the same information as the errorhndlr routine, but does not cause an exit from the mvsrxpli.rxp REXX command file.
11. Allocate data sets that are used by the CICS translator. The data sets allocated here correspond to the data sets that are used in your current JCL for the CICS translator.
12. If the DB2 precompiler was run, reallocate the file containing the expanded PL/I source from the DB2 precompiler to the file where the CICS translator expects the PL/I source to be located. If the DB2 precompiler was not run, allocate the file where the CICS translator expects the PL/I source to be located to the input being passed from TeamConnection.
13. Invoke the CICS precompiler, specifying the parameters to be used. The parameters are hard-coded in this example.
14. If there was an error, free the output data set where the PL/I source that was expanded by the CICS translator is normally placed. Also call the errorhndlr routine to format the CICS translator messages to go back to TeamConnection. The errorhndlr routine exits from the mvsrxpli.rxp REXX command file.
15. If there were no errors, call the TCOUThndlr routine to save the messages and other information that were written to the SYSPRINT file by the CICS translator. The TCOUThndlr routine saves the same information as the errorhndlr routine, but does not cause an exit from the mvsrxpli.rxp REXX command file.
16. If the CICS translator was run, reallocate the file containing the expanded PL/I source from the CICS translator to the file where the PL/I compiler expects the PL/I source to be located. If the CICS was not run and the DB2 precompiler was run, reallocate the file containing the expanded PL/I source from the DB2 precompiler to the file where the PL/I compiler expects the PL/I source to be located. If neither the PL/I translator nor the DB2 precompiler were run, allocate the file where the PL/I compiler expects the PLI source to be located to the input being passed from TeamConnection.
17. Allocate the remaining data sets that are used by the PL/I compiler. The data sets allocated here correspond to the data sets that are used in your current JCL for the PL/I compiler.

18. Allocate the data set that contains the include members that are maintained on TeamConnection.
19. Invoke the PL/I compiler, specifying the parameters to be used. The parameters are hard-coded in this example. Note that different parameters can be used depending on whether the CICS translator was run or not.
20. If there was an error, call the errorhndlr routine to format the PL/I compiler messages to go back to TeamConnection. The errorhndlr routine exits from the mvsrxpli.rxp REXX command file.
21. If there were no errors, call the TCOUThndlr routine to save the messages and other information that were written to the SYSPRINT file by the PL/I compiler. The TCOUThndlr routine saves the same information as the errorhndlr routine, but does not cause an exit from the mvsrxpli.rxp REXX command file.
22. Use the OUTLIST DD statement as a way of determining the output parts. Because &TCOUTPUT in the IRXJCL job shown in Figure 6 on page 9 removes the file extension from the output parts, this technique relies on having naming conventions such that the DBRM output part and the object module have the same part name. For example, plidb2.dbr and plidb2.obj are the names of the DBRM and the object module respectively.
23. The TCOUThndlr routine can be called whenever there is information that you want to save from a step in the REXX command file. In mvsrxpli.rxp, TCOUThndlr is called to save the information from the DB2 precompiler, CICS translator, and PL/I compiler steps. It is identical to the errorhndlr routine, except that TCOUThndlr returns to the invoking routine, while the errorhndlr exits from mvsrxpli.rxp.
24. The errorhndlr routine is only called when an unacceptable return code has occurred in the precompile, translate, compile, and so on. It copies the error information from a file such as the compiler's SYSPRINT to the temporary data set associated with the ERRLIST DD statement in the IRXJCL job shown in Figure 6 on page 9. Because the ERRLIST DD statement includes the parameter TCEXT=TCOUT, it causes any output placed in this data set to be sent back to TeamConnection and to appear in the **View build message** for the part that was built.

SAMPLE PARTS AND BUILD TREES FOR THE BASIC TECHNIQUE

This section contains sample parts and their corresponding build trees.

The build trees are shown with the following assumptions:

mvsrxpli.rxp (the REXX command file for DB2 precompile, CICS translate, and PL/I compile shown in Figure 8 on page 11) **is not** stored on the MVS host. Therefore it is necessary to include mvsrxpli.rxp as input to the .obj part.

mvsrplnk.rxp (the REXX command file for link edit and DB2 bind shown in Figure 29 on page 74) **is** stored on the MVS host. Therefore mvsrplnk.rxp must not be included as input to the .lkd part.

A REXX command file can be used to create any of the build trees shown below. See Appendix B, “SAMPLE REXX TO CREATE BUILD TREES” on page 91 for a sample REXX command file.

SIMPLE PL/I PROGRAM INCLUDE MEMBER

```
DCL 1 HDR,
        5  ASA         CHAR(1)       INIT(' '),
         5 OUTDATA CHAR(132)    INIT('TEAMCONNECTION INCLUDE STATEMENT');
```

Figure 9. PLIPGM.INC - PL/I Include Member

© Copyright IBM Corp. 1997

PL/I PROGRAM, LINK EDIT STATEMENTS, AND BUILD TREE

```
PLIPGM: PROC OPTIONS(MAIN);
 /                                                    /
  / Simple program for testing PL/I compiler        /
 /                                                    /
 %INCLUDE  PLIPGM;

  PUT LIST('PLIPGM STARTS');
  PUT LIST('PLIPGM ENDS');
END;
```

Figure 10. PLIPGM.PLI - PL/I Program

```
 INCLUDE  OBJECT(PLIPGM)
 NAME  PLIPGM(R)
```

Figure 11. PLIPGM.LNK - PL/I Link Edit Statements

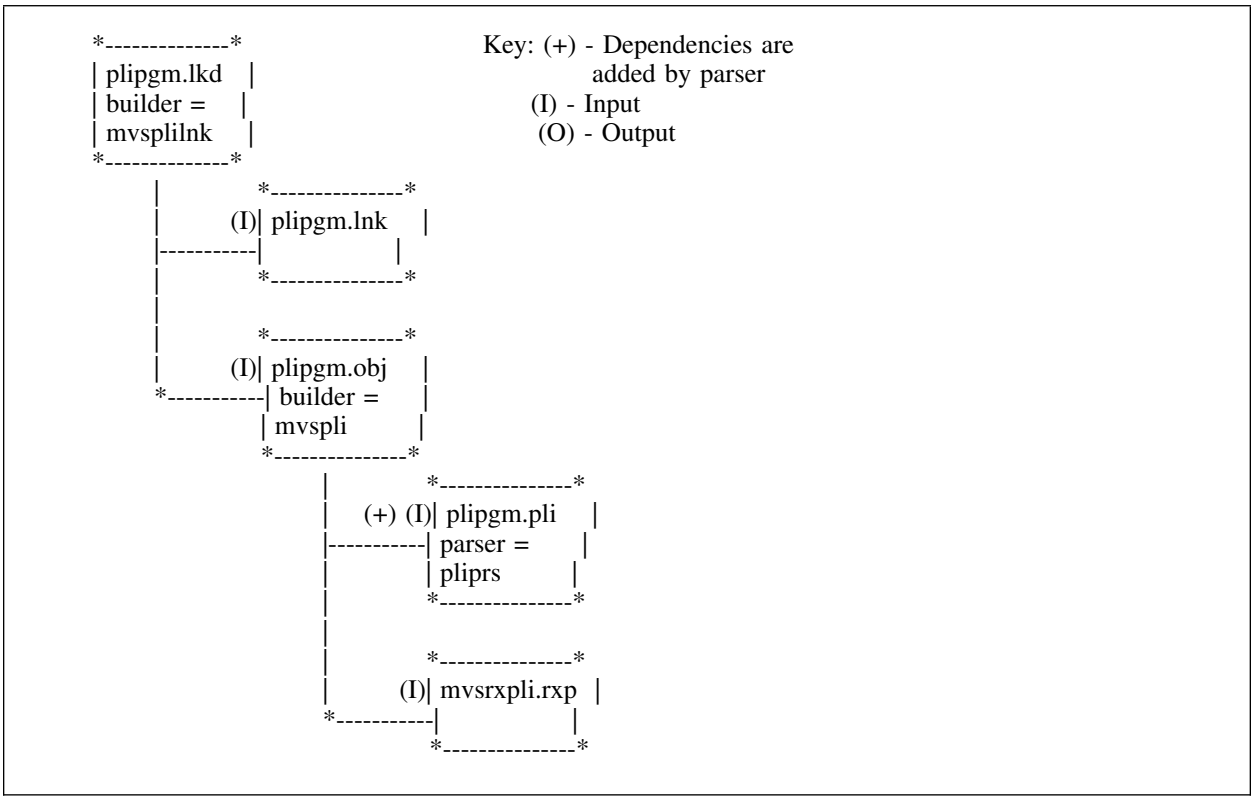

Figure 12. PLIPGM - Build Tree for PL/I Program

```
teamc Part   -connect plipgm.pli      -type TCPart -release release1 -workarea initwa
               -parent plipgm.obj -input
teamc Part   -connect mvsrxpli.rxp -type TCPart -release release1 -workarea initwa
               -parent plipgm.obj -input
teamc Part   -connect plipgm.obj      -type TCPart -release release1 -workarea initwa
               -parent plipgm.lkd -input
teamc Part   -connect plipgm.lnk      -type TCPart -release release1 -workarea initwa
               -parent plipgm.lkd -input
```

Figure 13. PLIPGM - Connect Commands to Create Build Tree

PL/I WITH DB2 PROGRAM, LINK EDIT STATEMENTS, AND BUILD TREE

```
PLIDB2: PROC OPTIONS(MAIN);
 /                                                          /
  / Simple program for testing DB2 precompiler      /
 /                                                          /
 %INCLUDE  PLIPGM;

            EXEC  SQL
                DECLARE  EZE  1  1WDT15A
                  CURSOR  FOR
                  SELECT
                    CURRENT  TIMESTAMP
                  FROM
                    TURBO.MESSAGE  T1
            ;

  PUT LIST('PLIDB2 STARTS');
  PUT LIST('PLIDB2 ENDS');
END;
```

Figure 14. PLIDB2 - PL/I Program with DB2

```
 INCLUDE  OBJECT(PLIDB2)
 INCLUDE  SYSLIB(DSNELI)
 NAME  PLIDB2(R)
```

Figure 15. PLIDB2.LNK - PL/I with DB2 Link Edit Statements

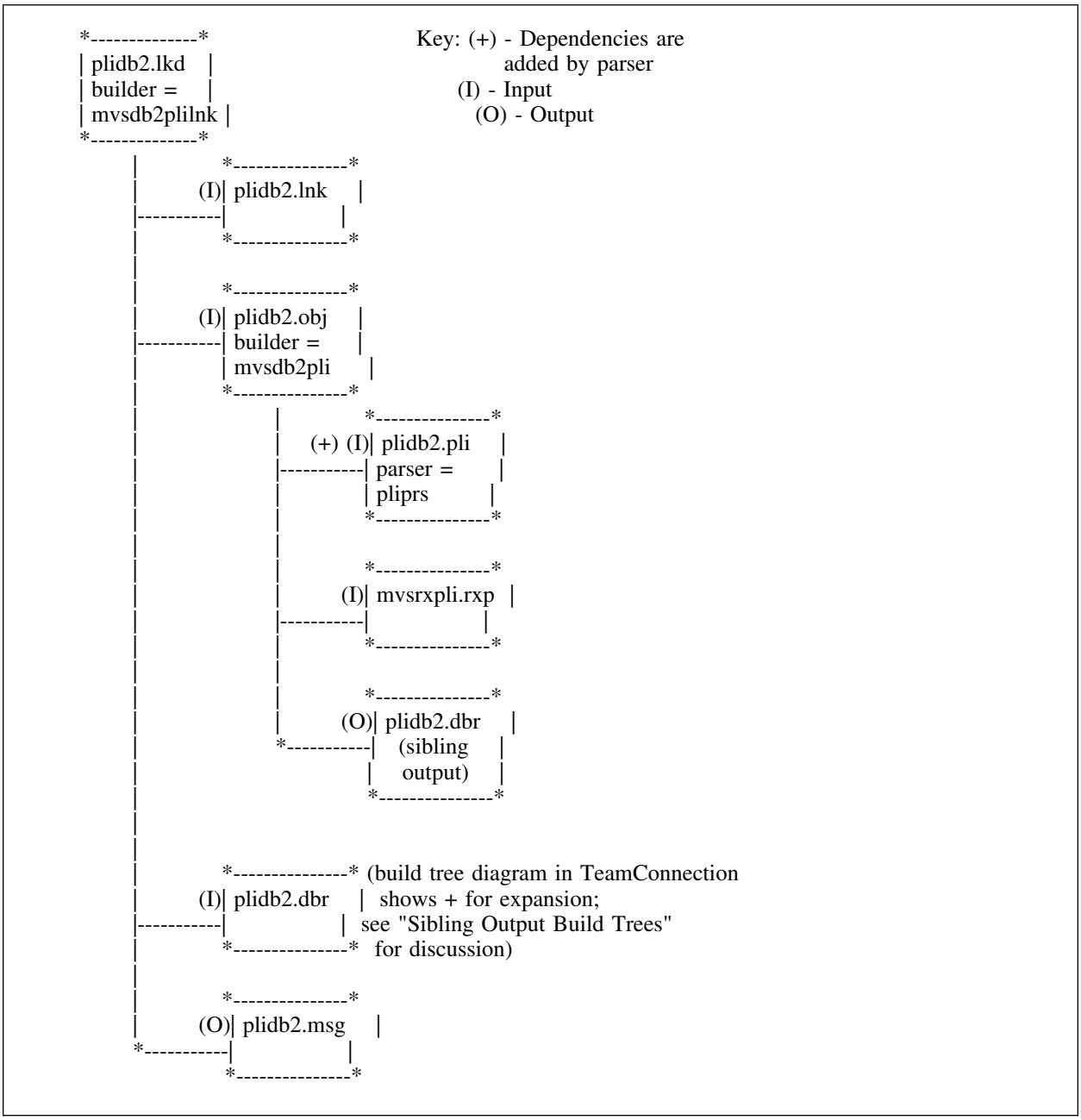

Figure 16. PLIDB2 - Build Tree for PL/I Program with DB2

```
teamc Part   -connect plidb2.pli      -type TCPart -release release1 -workarea initwa
               -parent plidb2.obj -input
teamc Part   -connect mvsrxpli.rxp -type TCPart -release release1 -workarea initwa
               -parent plidb2.obj -input
teamc Part   -connect plidb2.dbr      -type TCPart -release release1 -workarea initwa
               -parent plidb2.obj -output
teamc Part   -connect plidb2.dbr      -type TCPart -release release1 -workarea initwa
               -parent plidb2.lkd -input
teamc Part   -connect plidb2.obj      -type TCPart -release release1 -workarea initwa
               -parent plidb2.lkd -input
teamc Part   -connect plidb2.lnk      -type TCPart -release release1 -workarea initwa
               -parent plidb2.lkd -input
teamc Part   -connect plidb2.msg      -type TCPart -release release1 -workarea initwa
               -parent plidb2.lkd -output
```

Figure 17. PLIDB2 - Connect Commands to Create Build Tree

The DBRM member (plidb2.dbr) is output from the build of the object module and input to the build for the load module. This insures that the link edit and bind are done on the same MVS host and at the same time to avoid a mismatch in the date/time stamp for the load module and DB2 package.

The message member (plidb2.msg) is output from the build of the load module. It is used to assist in determining the number of .obj / .dbr pairs in the build. This is done by matching the .msg part with the .lnk part. For further details, see Appendix A, “BUILDER, BUILD SCRIPT, AND REXX FOR LINK EDIT AND BIND” on page 69.

PL/I WITH CICS PROGRAM, LINK EDIT STATEMENTS, AND BUILD TREE

```
PLICICS: PROC OPTIONS(MAIN);
 /                                                          /
  /  Simple program for testing CICS translator       /
 /                                                          /
  %INCLUDE  PLIPGM;
   DCL EYE_CATCHER CHAR(5 ) INIT('PLICICS PROGRAM');
   DCL TIME_INFO FIXED DEC(15);
   EXEC CICS ASKTIME ABSTIME(TIME_INFO);
END;
```

Figure 18. PLICICS - PL/I Program with CICS

```
 INCLUDE  SYSLIB(DFHPL1OI)
 REPLACE  PLISTART
 INCLUDE  OBJECT(PLICICS)
 NAME  PLICICS(R)
```

Figure 19. PLICICS.LNK - PL/I with CICS Link Edit Statements

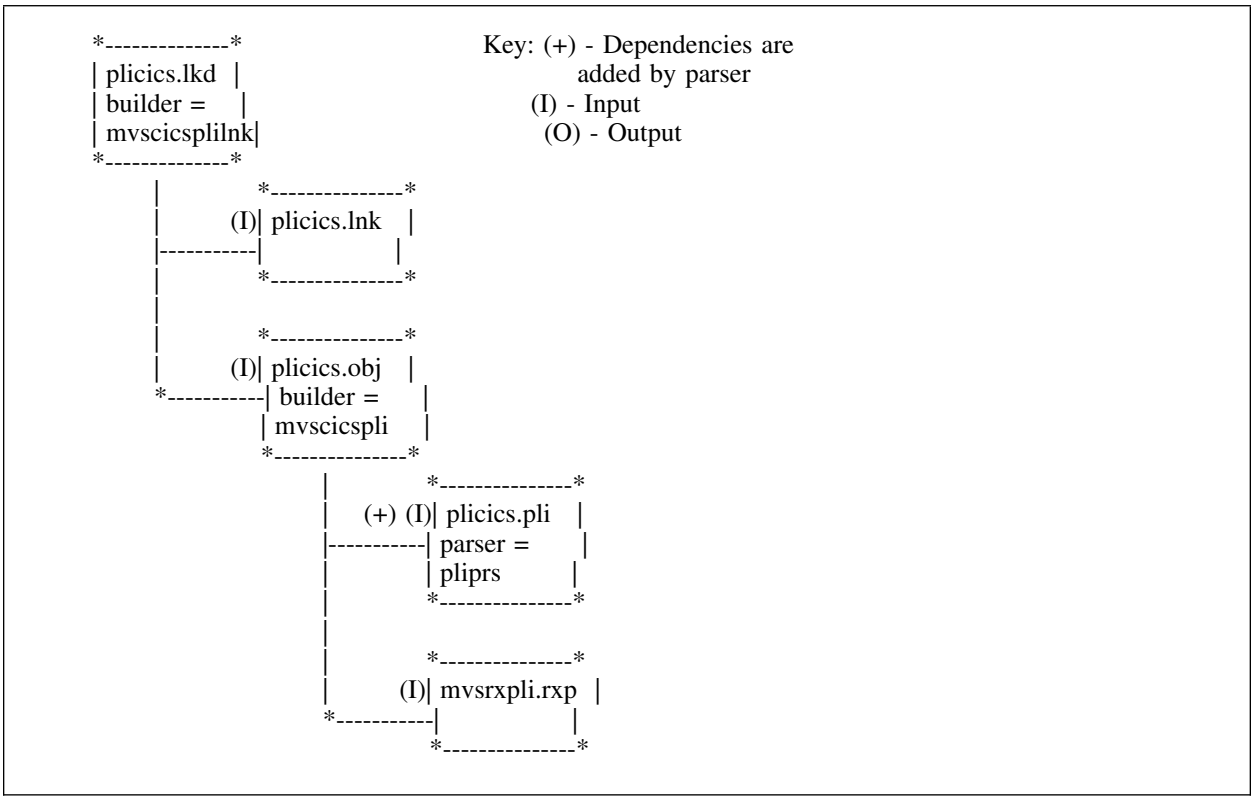

Figure 20. PLICICS - Build Tree for PL/I Program with CICS

```
teamc Part   -connect plicics.pli    -type TCPart -release release1 -workarea initwa
               -parent plicics.obj -input
teamc Part   -connect mvsrxpli.rxp -type TCPart -release release1 -workarea initwa
               -parent plicics.obj -input
teamc Part   -connect plicics.obj    -type TCPart -release release1 -workarea initwa
               -parent plicics.lkd -input
teamc Part   -connect plicics.lnk    -type TCPart -release release1 -workarea initwa
               -parent plicics.lkd -input
```

Figure 21. PLICICS - Connect Commands to Create Build Tree

PL/I WITH DB2 AND CICS PROGRAM, LINK EDIT STATEMENTS, AND BUILD TREE

```
PLIDCIC: PROC OPTIONS(MAIN);
 /                                                          /
  / Simple program for testing DB2 precompiler      /
  / and CICS translator                                 /
 /                                                          /
 %INCLUDE  PLIPGM;
  DCL EYE_CATCHER CHAR(5 ) INIT('PLIDCIC PROGRAM');
  DCL TIME_INFO FIXED DEC(15);

            EXEC  SQL
                DECLARE  EZE  1  1WDT15A
                  CURSOR  FOR
                  SELECT
                    CURRENT  TIMESTAMP
                  FROM
                    TURBO.MESSAGE  T1
            ;

  EXEC CICS ASKTIME ABSTIME(TIME_INFO);

END;
```

Figure 22. PLIDCIC - PL/I Program with DB2 and CICS

```
 INCLUDE  SYSLIB(DFHPL1OI)
 REPLACE  PLISTART
 INCLUDE  OBJECT(PLIDCIC)
 INCLUDE  SYSLIB(DSNCLI)
 NAME  PLIDCIC(R)
```

Figure 23. PLIDCIC.LNK - PL/I with DB2 and CICS Link Edit Statements

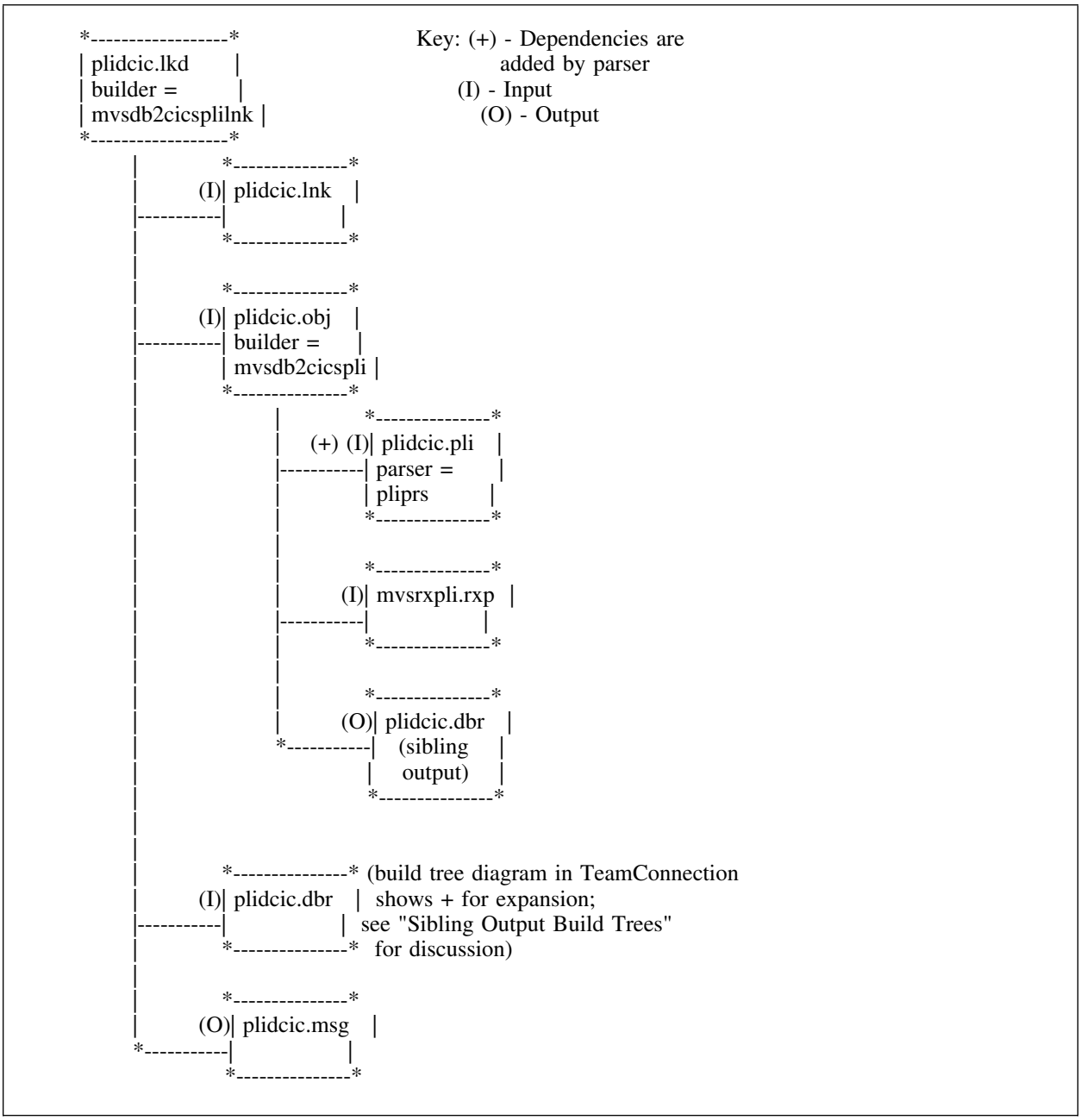

Figure 24. PLIDCIC - Build Tree for PL/I Program with DB2 and CICS

```
teamc Part   -connect plidcic.pli -type TCPart -release release1 -worka rea initwa
               -parent plidcic.obj -input
teamc Part   -connect mvsrxpli.rxp -type TCPart -release release1 -workarea initwa
               -parent plidcic.obj -input
teamc Part   -connect plidcic.dbr -type TCPart -release release1 -worka rea initwa
               -parent plidcic.obj -output
teamc Part   -connect plidcic.dbr -type TCPart -release release1 -worka rea initwa
               -parent plidcic.lkd -input
teamc Part   -connect plidcic.obj -type TCPart -release release1 -worka rea initwa
               -parent plidcic.lkd -input
teamc Part   -connect plidcic.lnk -type TCPart -release release1 -worka rea initwa
               -parent plidcic.lkd -input
teamc Part   -connect plidcic.msg -type TCPart -release release1 -worka rea initwa
               -parent plidcic.lkd -output
```

Figure 25. PLIDCIC - Connect Commands to Create Build Tree

The DBRM member (plidcic.dbr) is output from the build of the object module and input to the build for the load module. This insures that the link edit and bind are done on the same MVS host and at the same time to avoid a mismatch in the date/time stamp for the load module and DB2 package.

The message member (plidcic.msg) is output from the build of the load module. It is used to assist in determining the number of .obj / .dbr pairs in the build. This is done by matching the .msg part with the .lnk part. For further details, see Appendix A, “BUILDER, BUILD SCRIPT, AND REXX FOR LINK EDIT AND BIND” on page 69.

SIBLING OUTPUT BUILD TREES

As shown in Figure 16 on page 27 and Figure 24 on page 32, you can connect additional output part(s) to a part that will be built. For example, in Figure 16 on page 27:

plidb2.dbr is an output of plidb2.obj.

plidb2.msg is an output of plidb2.lkd.

In TeamConnection terms, plidb2.dbr and plidb2.obj are **sibling outputs**. Similarly, plidb2.msg and plidb2.lkd are sibling outputs.

Sibling outputs have the following effects on part properties:

When a builder is assigned or changed for one part, the same change occurs for both parts.

Any parameters specified for the build of one part are used for building both parts.

Sibling outputs have the following effects on the build tree diagram:

Expansion of either sibling output part shows the other sibling as an output. For example, the build trees for plidb2.obj and plidb2.dbr are shown in Figure 26.

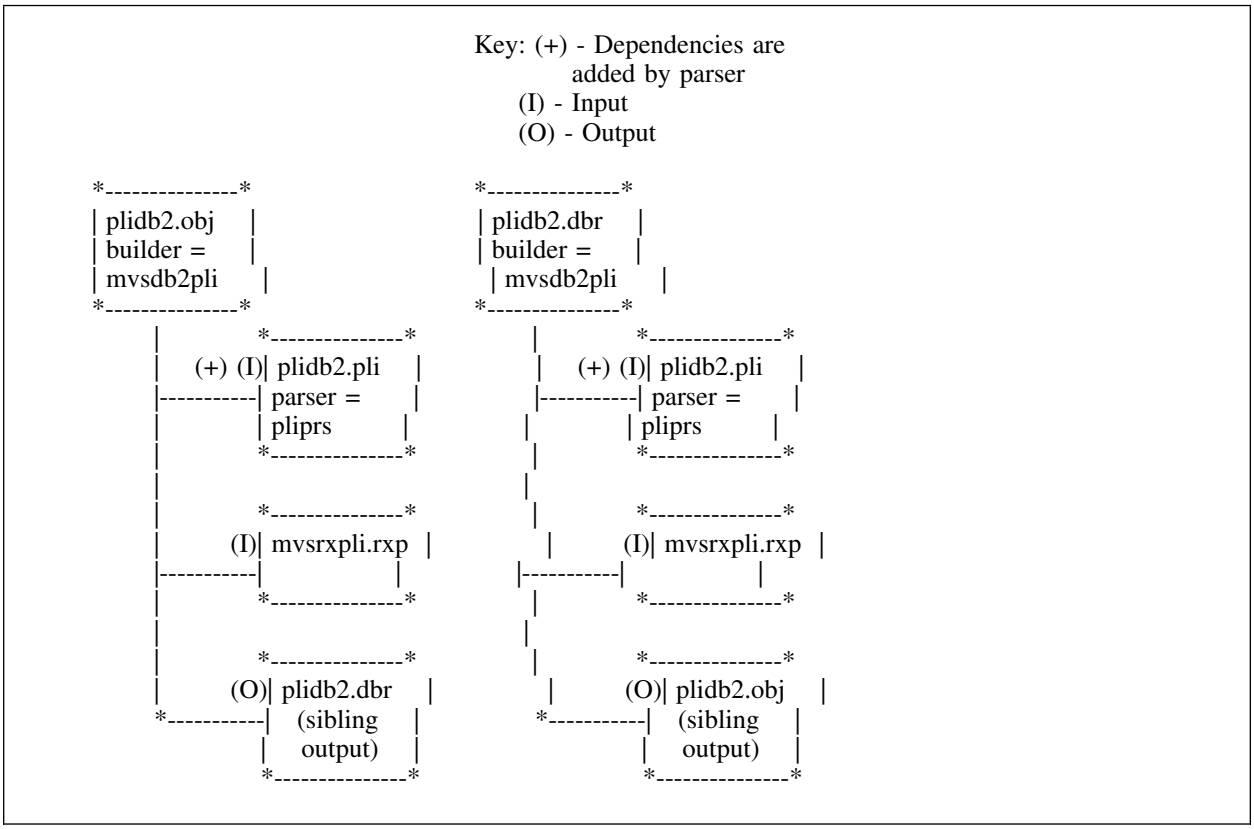

Figure 26. Build Trees for Sibling Outputs PLIDB2.OBJ and PLIDB2.DBR

When the sibling is shown as an output part, there is no part expansion in the TeamConnection build tree diagram. For example, in Figure 16 on page 27, plidb2.msg only appears as an output part. The TeamConnection build tree diagram does not include a **+** to indicate an expansion is possible. Similarly, where plidb2.dbr is an output of plidb2.obj, the TeamConnection build tree diagram does not include the **+**.

To see one sibling (plidb2.dbr) as output of the other in the build tree diagram, you must first expand its sibling output (plidb2.obj). Thus there is no need to expand the output part (plidb2.dbr) to see the relationship between the two parts. Doing an expansion of plidb2.dbr would take you in a circle.

When a sibling output of one part is shown as an input to another part, there is support for a part expansion in the TeamConnection build tree diagram. For example, when plidb2.dbr is an input to plidb2.lkd, the TeamConnection build tree diagram includes the **+** to indicate plidb2.dbr can be expanded.

To see the input part (plidb2.dbr) in the build tree diagram, you must first expand the part that is built (plidb2.lkd). You might want to see the expanded build tree for the input part (plidb2.dbr) because its sibling output might not be part of the same build tree. In Figure 16 on page 27, both sibling outputs (plidb2.obj and plidb2.dbr) are inputs to the final part (plidb2.lkd). In this case, either sibling or both can be expanded. To simplify the diagrams shown in Figure 12 on page 25 through Figure 24 on page 32, only the .obj part is expanded.

PUTTING IT ALL TOGETHER

The following sections show the TeamConnection processing to do builds with the REXX command file.

CREATE A NEW FAMILY

This section shows how to create a new family, update the hosts and services files, and start the family.

Action	Description
Make sure the following are running: TCP/IP ObjectStore Make a directory called e:\family.	
Open the TeamConnection group folder.	The "TeamConnection Group - Icon View" window appears.
Double-click on the TeamConnection Family Administrator icon.	The "TeamConnection Family Administrator" window appears.
Select **Family**, then **New**, and then **Default**.	An "Untitled-Settings" notebook appears, with the "Family Information" page displayed.
Enter the following: Family Information: **Family name** plirexx **Path** e:\family **Port** 9807 or some other unused port **Mailer** mailexit or null.cmd	
Select the "Initial Superuser" tab.	The "Initial Superuser" page appears.
Enter the following: Initial Superuser: **Login** SuprUser **Name** Super User **Host** SuprHost **User ID** <leave blank> **Password** <leave blank>	

© Copyright IBM Corp. 1997

Action	Description
Below the notebook, press the **Create** push button.	The "Create Family" window appears and runs the commands to create the family and the super user. This takes about 5 minutes. The return code at the end should be 0.
Press the **OK** push button.	The "TeamConnection Family Administrator" window appears with an icon for plirexx in the window.
From an OS/2 window, enter: `echo %ETC%`	The current value of the ETC environment variable appears. This value is the name of the directory to use in the next few steps. The instructions below assume that the directory is C:\MPTN\ETC.
Change the directory and drive to C:\MPTN\ETC.	
Add plirexx, mvssock, and tsosock to C:\MPTN\ETC\hosts. Also make sure that there is a carriage return after each line. The hosts listed in Table 1 on page 39 should be in the hosts file.	The sockets are used as follows: **plirexx** Family Server **mvssock** MVS Build processor using RUNPGM **tsosock** MVS Build processor using RUNPGMT
Add plirexx, mvssock, and tsosock to C:\MPTN\ETC\services. The services listed in Table 2 on page 39 should be in the services file.	
From the "TeamConnection Family Administrator" window, double-click on the plirexx icon.	The "plirexx - Family Servers" window appears.
Press the **Start** push button in the "Family Server" section of the window.	The family server is started. When the message to "Press CTRL-C to stop" appears, minimize the "plirexx - Family Servers" window and the "TeamConnection Family Administrator" window.

Table 1. Hosts File		
IP Address	**Host Name**	**Family / Socket**
9.37.195.42	FamHost.raleigh.ibm.com	plirexx
9.67.43.25	carmvs1.raleigh.ibm.com	mvssock
9.67.43.25	carmvs1.raleigh.ibm.com	tsosock

Table 2. Services File		
Family / Socket	**Port Number**	**Comment**
plirexx	9807/tcp	#port address for plirexx
mvssock	32901/tcp	#port address for MVS build processor using RUNPGM
tsosock	32911/tcp	#port address for MVS build processor using RUNPGMT

START THE TEAMCONNECTION CLIENT

This section shows how to start the TeamConnection client and change the **Settings**.

Action	Description
On the TeamConnection server machine, create a working directory called E:\WORK\plirexx	
Open the TeamConnection Group folder.	
Select the TeamConnection client icon with mouse button 2.	A list of options appears.
Select **Settings**.	The "TeamConnection Client - Settings" notebook appears.
On the "Program" page, change the working directory to E:\WORK\plirexx	
Close the "TeamConnection Client - Settings" notebook.	
Double-click on the TeamConnection client icon.	The "TeamConnection - Tasks" window appears.
Select **Windows** and then **Settings**.	The "Settings" notebook appears, with the "Environment" page showing.
Enter the following on the "Environment" page: **Family** plirexx **Release** release1 **Become user** SuprUser **User ID** SuprUser **Relative directory** e:\mytemp Then press the **Apply** push button.	
Press the **Test Connection to Server** push button.	The "Settings" information window should appear and indicate that you connected to the server. There should be no error messages.
Press **OK** to close the information window	
Select the **Pool** tab in the "Settings" notebook. Enter **pool1** for the **Pool** value. Press the **Apply** push button.	All build activity in the basic demonstration uses "pool1". This value is the default and is automatically entered into the "Pool" field of the build windows.

Action	Description
Select the **GUI** tab in the "Settings" notebook. Make sure that the checkbox for **Verbose commands** is deselected. Press the **Apply** push button.	Deselecting Verbose commands reduces the number of messages you receive.
Double-click on the upper left corner of the "Settings" window.	The "TeamConnection - Tasks" window appears. The user ID and plirexx should be in the lower left corner of the tasks window.

CREATE COMPONENTS

This section creates the component structure for the demonstration. The demonstration uses the overall controlling component projname. The other components can be used for modifications of the demonstration to show activities related to components.

Action	Description
Select **Actions**, then **Components**, and then **Create**.	The "Create Components" window appears.
Enter the components shown in Table 3, pressing the **Apply** push button after each entry.	
Press the **Cancel** push button.	The "TeamConnection - Tasks" window appears.
Select **Objects**, then **Components**, and then **Components**.	The "Components Filter" window appears.
In the "History" section, select **Show all Components**. Press the **OK** push button.	The "TeamConnection - Components" window appears.
Close the "TeamConnection - Components" window.	The "TeamConnection - Tasks" window appears.

Table 3. Components

Component	Parent	Process	Owner	Description
projname	root	prototype	SuprUser	overall controlling component
subsys1	projname	prototype	SuprUser	subsystem 1
subsys2	projname	prototype	SuprUser	subsystem 2

CREATE A RELEASE

This section shows how to create a release. Parts, work areas, and builders cannot be created without first specifying a release.

Action	Description
Select **Actions**, then **Releases**, and then **Create**.	The "Create Releases" window appears.
Enter the following information: **Releases** release1 **Component** projname **Process** no_track **Owner** SuprUser **Description** projname release 1 Also select **Automatic version pruning**. Press the **OK** push button.	The "TeamConnection - Tasks" window appears. The release has been created.

CREATE BUILDERS

This section creates the builders that are used throughout the demonstration. All builders should be created at this time so that you do not need to return to this activity later.

The builders for link edit and bind (mvsplilnk, mvsdb2plilnk, mvscicsplilnk, and mvsdb2cicsplilnk) are described in Appendix A, “BUILDER, BUILD SCRIPT, AND REXX FOR LINK EDIT AND BIND” on page 69.

Action	Description
Select **Actions**, then **Builders**, and then **Create** on client workstation.	The "Create Builders" window appears.
Enter the builder information shown in Figure 27 on page 43, pressing the **Apply** push button after each entry.	A NULL builder could be used for a collector part in the demonstration.
Press the **Cancel** push button to return to the Tasks list.	The "TeamConnection - Tasks" window appears.
Select **Objects** and then **Builders**.	The "Builder Filter" window appears.
In the "History" section, select **Show all Builders**. Press the **OK** push button.	The "TeamConnection - Builders" window appears.
Close the "TeamConnection - Builders" window.	The "TeamConnection - Tasks" window appears.

```
Builder          mvspli
Release          release1
Script           mvspli.jcl
Environment      mvstso
Comp. oper.      <=
RC value         4
File type        text
Source file      e:\tcdemo2\plirexx\scripts\mvspli.jcl
 Parameters      'MVSRXPLI N N Y'
Timeout          1

Builder          mvsdb2pli
Release          release1
Script           mvspli.jcl
Environment      mvstso
Comp. oper.      <=
RC value         4
File type        text
Source file      e:\tcdemo2\plirexx\scripts\mvspli.jcl
 Parameters      'MVSRXPLI Y N Y'
Timeout          1

Builder          mvscicspli
Release          release1
Script           mvspli.jcl
Environment      mvstso
Comp. oper.      <=
RC value         4
File type        text
Source file      e:\tcdemo2\plirexx\scripts\mvspli.jcl
 Parameters      'MVSRXPLI N Y Y'
Timeout          1

Builder          mvsdb2cicspli
Release          release1
Script           mvspli.jcl
Environment      mvstso
Comp. oper.      <=
RC value         4
File type        text
Source file      e:\tcdemo2\plirexx\scripts\mvspli.jcl
 Parameters      'MVSRXPLI Y Y Y'
Timeout          1
```

Figure 27 (Part 1 of 3). Builder Information

```
Builder          mvsplilnk
Release          release1
Script           mvslnk.jcl
Environment      mvstso
Comp. oper.      ==
RC value
File type        none
Source file
 Parameters      'MVSRPLNK N N N NA NA &TCWKAREA'
Timeout          4

Builder          mvsdb2plilnk
Release          release1
Script           mvslnk.jcl
Environment      mvstso
Comp. oper.      ==
RC value
File type        none
Source file
 Parameters      'MVSRPLNK Y N N DSNA TOOLCICS &TCWKAREA'
Timeout          4

Builder          mvscicsplilnk
Release          release1
Script           mvslnk.jcl
Environment      mvstso
Comp. oper.      ==
RC value
File type        none
Source file
 Parameters      'MVSRPLNK N N Y NA NA &TCWKAREA'
Timeout          4

Builder          mvsdb2cicsplilnk
Release          release1
Script           mvslnk.jcl
Environment      mvstso
Comp. oper.      ==
RC value
File type        none
Source file
 Parameters      'MVSRPLNK Y N Y DSNA TOOLCICS &TCWKAREA'
Timeout          4
```

Figure 27 (Part 2 of 3). Builder Information

```
Builder          mvsnull
Release          release1
Script           NULL
Environment      mvstso
Comp. oper.      ==
RC value
File type        none
Source file
Parameters
Timeout          1
```

Figure 27 (Part 3 of 3). Builder Information

CREATE A PARSER

This section creates parsers to determine if a part has a dependency. The term "create a parser" involves telling TeamConnection which command to run for parsing a part. Programs that actually get control to do the parsing must be written just like any other tool.

Action	Description
Select **Actions**, then **Parsers**, and then **Create**.	The "Create Parser" window appears.
Set the following fields: **Parser** pliprs **Release** release1 **Command** fhbplprs.cmd Press the **OK** push button. **Note:** fhbplprs.cmd is a sample PL/I parser supplied with TeamConnection.	The "TeamConnection - Tasks" window appears. The parser has been created.

OPEN A FEATURE AND CREATE A WORK AREA

This section opens a feature and creates a work area for the initial load of the database. This is a typical activity for migrating existing parts (source code, tables) into TeamConnection.

Action	Description
Select **Actions**, then **Features**, and then **Open**.	The "Open Feature" window appears.
Enter the following information: **Component** projname **Remarks** initial load of the family **Name** initialize **Prefix** f Press the **OK** push button.	The "Open Feature" information window appears with a message that the new feature was opened successfully.
Press the **OK** push button again to return to the "TeamConnection - Tasks" window.	
Select **Actions**, then **Features**, and then **Accept**.	The "Accept Feature" window appears.
Enter the following information: **Features** initialize **Remarks** accepting initial load of the family Press the **OK** push button.	The "TeamConnection - Tasks" window appears. The feature has been accepted.
Select **Actions**, then **Work areas**, and then **Create**.	The "Create Work Areas" window appears.
Enter the following information: **Work areas** initwa **Releases** release1 **Owner** SuprUser Press the **OK** push button.	The "TeamConnection - Tasks" window appears. The work area has been created.

CREATE PARTS

This section creates the parts for the initial load of the database as well as the output parts that will be created from builds.

Action	Description
Select **Actions**, then **Parts**, and then **Create**.	The "Create Parts" window appears.
Set the following fields for each part: **Part names** see Table 4 on page 48 **Release** release1 **Work area** initwa **Component** projname **File type** see Table 4 on page 48 **Source** see Table 4 on page 48 **Source file** see Table 4 on page 48 **Builder** see Table 4 on page 48 **Parser** see Table 4 on page 48 Enter the parts shown in Table 4 on page 48, pressing the **Apply** push button after each entry.	The extension of include parts can be "inc" in lower case or upper case. The case of the part name without the extension must match the case of the copybook in the source program. To set the **File type** to **None** for the .lkd parts, you might need to set **Source** to **Same** first.
Press the **Cancel** push button to return to the Tasks list.	The "TeamConnection - Tasks" window appears.

Table 4 (Page 1 of 2). projname Create Part Information					
Part name	**File type**	**Source**	**Source file**	**Builder**	**Parser**
mvsrxpli.rxp	Text	Copy from	e:\tcdemo2\plirexx \parts\mvsrxpli.rxp		
PLIPGM.inc	Text	Copy from	e:\tcdemo2\plirexx \parts\plipgm.inc		pliprs
PL/I Only					
plipgm.pli	Text	Copy from	e:\tcdemo2\plirexx \parts\plipgm.pli		pliprs
plipgm.obj	Binary	No source		mvspli	
plipgm.lnk	Text	Copy from	e:\tcdemo2\plirexx \parts\plipgm.lnk		
plipgm.lkd	None			mvsplilnk	
PL/I with DB2					
plidb2.pli	Text	Copy from	e:\tcdemo2\plirexx \parts\plidb2.pli		pliprs
plidb2.obj	Binary	No source		mvsdb2pli	
plidb2.dbr	Binary	No source			
plidb2.lnk	Text	Copy from	e:\tcdemo2\plirexx \parts\plidb2.lnk		
plidb2.msg	Text	No source			
plidb2.lkd	None			mvsdb2plilnk	
PL/I with CICS					
plicics.pli	Text	Copy from	e:\tcdemo2\plirexx \parts\plicics.pli		pliprs
plicics.obj	Binary	No source		mvscicspli	
plicics.lnk	Text	Copy from	e:\tcdemo2\plirexx \parts\plicics.lnk		
plicics.lkd	None			mvscicsplilnk	

PL/I with DB2 and CICS					
plidcic.pli	Text	Copy from	e:\tcdemo2\plirexx \parts\plidcic.pli		pliprs
plidcic.obj	Binary	No source		mvsdb2cicspli	
plidcic.dbr	Binary	No source			
plidcic.lnk	Text	Copy from	e:\tcdemo2\plirexx \parts\plidcic.lnk		
plidcic.msg	Text	No source			
plidcic.lkd	None			mvsdb2cicsplilnk	

CREATE BUILD TREE

This section shows how to create a build tree. A build tree shows the configuration of the application. TeamConnection uses the build tree to determine what parts need to be built.

Action	Description
Select **Objects**, then **Parts**, and then **Parts**.	The "Part Filter" window appears.
Set the following fields: **Release** release1 **Work area** initwa In the "History" section, select **Show all Parts** Press the **OK** push button.	The "TeamConnection - Parts" window appears.
For each icon in Table 5 on page 52: select the **icon** in the "TeamConnection - Parts" window click on the second mouse button select **Connect** from the pop up Set the following fields and radio buttons in the "Connect Parts" window: **Parent** see Table 5 on page 52 **Relation to parent** see Table 5 on page 52 Press the **OK** push button.	The fields "Path names", "Type", "Release", and "Work area" should already have the correct values.
Close the "TeamConnection - Parts" window.	The "TeamConnection - Tasks" window appears.
Select **Objects**, then **Parts**, and then **BuildView**.	The "BuildView Filter" window appears.
Set the following fields: **Release** release1 **Work area** initwa **Base names** like %.obj and sort 1st Asc Press the **OK** push button.	The "TeamConnection - BuildView" window appears.
Expand the **plipgm.obj** icon to show the entire tree.	Examine the tree for proper structure (see Figure 12 on page 25).

Action	Description
Also expand the following .obj parts and compare them to the expected build tree diagrams. **plidb2** Figure 16 on page 27 **plicics** Figure 20 on page 30 **plidcic** Figure 24 on page 32	

Table 5. release1 Build Tree Information		
Icon	**Parent**	**Relation**
PL/I Only		
plipgm.pli	plipgm.obj	input
mvsrxpli.rxp	plipgm.obj	input
plipgm.obj	plipgm.lkd	input
plipgm.lnk	plipgm.lkd	input
PL/I with DB2		
plidb2.pli	plidb2.obj	input
mvsrxpli.rxp	plidb2.obj	input
plidb2.dbr	plidb2.obj	output
plidb2.dbr	plidb2.lkd	input
plidb2.obj	plidb2.lkd	input
plidb2.lnk	plidb2.lkd	input
plidb2.msg	plidb2.lkd	output
PL/I with CICS		
plicics.pli	plicics.obj	input
mvsrxpli.rxp	plicics.obj	input
plicics.obj	plicics.lkd	input
plicics.lnk	plicics.lkd	input
PL/I with DB2 and CICS		
plidcic.pli	plidcic.obj	input
mvsrxpli.rxp	plidcic.obj	input
plidcic.dbr	plidcic.obj	output
plidcic.dbr	plidcic.lkd	input
plidcic.obj	plidcic.lkd	input
plidcic.lnk	plidcic.lkd	input
plidcic.msg	plidcic.lkd	output

START THE MVS BUILD AGENT AND BUILD PROCESSOR

This section shows how to start an MVS build agent and build processor. Together this pair is called a build server. Build events wait in the job queue for a build agent to start.

Action	Description
Go to an OS/2 window (recommended on the machine that is running the TeamConnection family server).	
Make sure the TC_DBPATH environment variable is set to the proper value (e:\family\plirexx for example).	
Issue the command **teamagnt -f plirexx -e mvstso -p pool1 -s tsosock -k IBM-85**	The build agent starts in the OS/2 window.
Minimize the OS/2 window - DO NOT CLOSE.	
Go to an MVS window or terminal and logon.	
Submit the job that runs the TeamConnection RUNPGMT program (it might be in the data set logonid.TEAMC.JCLLOAD(RUNPGMT)). See Figure 4 on page 8 for an example of the JCL.	The job should be active on the MVS job queue and waiting for a file from the MVS build agent.
Logoff or monitor the job for errors.	

BUILD THE OBJECT MODULES

This section builds the MVS applications. **Build** puts build events on the job queue and the build agent removes them. The build agent sends the build script (from the builder) and the inputs to the build processor. The build processor runs the build script.

Action	Description
From the "TeamConnection - BuildView" window, select **plipgm.obj**.	
Select **Selected**, and then **Build**.	The "Build Parts" window appears.
Set the following fields and radio buttons: **Path name** plipgm.obj **Type** TCPart **Release** release1 **Work area** initwa **Pool** pool1 **Build mode** Normal **Parameters** **Detail file** Press the **OK** push button.	The "Build Progress" window appears.
Monitor the "Build Progress" window. Press the **OK** push button when the build is complete.	The "TeamConnection - BuildView" window appears.
Press F5 to refresh the "TeamConnection - BuildView" window.	The "TeamConnection - BuildView" window reappears. If the build was successful, plipgm.obj should have a green checkmark.
Select **plipgm.obj**	
Select **Selected**, then **View**, and then **View build message**.	The "View Build Message" window appears.
Press the **OK** push button.	The messages from the most recent build of plipgm.obj appear. Review the messages for any problems.
Optional - the MVS job can be cancelled with a print to examine the output, but would need to be submitted again for additional builds.	
Repeat the above steps to build the following parts: plidb2.obj plicics.obj plidcic.obj	

Action	Description
Close the "TeamConnection - BuildView" window.	The "TeamConnection - Tasks" window appears.

BUILD THE LOAD MODULES

Before you build the load modules, be sure that

mvslnk.jcl (the build script for link edit and bind, shown in Figure 28 on page 70) and

mvsrplnk.rxp (the REXX command file for link edit and bind, shown in Figure 29 on page 74)

are on the MVS host. Also be sure that:

The TEAMPROC DD statement in RUNPGMT points to the partitioned data set where you stored mvslnk.jcl

The SYSEXEC DD statement in mvslnk.jcl points to the partitioned data set where you stored mvsrplnk.rxp.

Action	Description
Select **Objects**, then **Parts**, and then **BuildView**.	The "BuildView Filter" window appears.
Set the following fields: **Release** release1 **Work area** initwa **Base names** like %.lkd and sort 1st Asc Press the **OK** push button.	The "TeamConnection - BuildView" window appears.
Expand the **plipgm.lkd** icon to show the entire tree.	Examine the tree for proper structure (see Figure 12 on page 25).
Also expand the following parts and compare them to the expected build tree diagrams. **plidb2** Figure 16 on page 27 **plicics** Figure 20 on page 30 **plidcic** Figure 24 on page 32	
From the "TeamConnection - BuildView" window, select **plipgm.lkd**.	
Select **Selected**, and then **Build**.	The "Build Parts" window appears.

Action	Description
Set the following fields and radio buttons: **Path name** plipgm.lkd **Type** TCPart **Release** release1 **Work area** initwa **Pool** pool1 **Build mode** Normal **Parameters** **Detail file** Press the **OK** push button.	The "Build Progress" window appears.
Monitor the "Build Progress" window. Press the **OK** push button when the build is complete.	The "TeamConnection - BuildView" window appears.
Press F5 to refresh the "TeamConnection - BuildView" window.	The "TeamConnection - BuildView" window reappears. If the build was successful, plipgm.lkd should have a green checkmark.
Select **plipgm.lkd**	
Select **Selected**, then **View**, and then **View build message**.	The "View Build Message" window appears.
Press the **OK** push button.	The messages from the most recent build of plipgm.lkd appear. Review the messages for any problems.
Optional - the MVS job can be cancelled with a print to examine the output, but would need to be submitted again for additional builds.	
Repeat the above steps to build the following parts: plidb2.lkd plicics.lkd plidcic.lkd	
Close the "TeamConnection - BuildView" window.	The "TeamConnection - Tasks" window appears.

VARIATIONS IN MVSPLI.JCL AND MVSRXPLI.RXP

The following sections describe variations in the basic technique that is described above.

"Keeping mvspli.jcl at the MVS Host"

"Keeping mvsrxpli.rxp at the MVS Host" on page 58

"Supporting Program Product Include Libraries" on page 58.

"Supporting Program Product Include Libraries Concatenated with TeamConnection Copybooks" on page 59.

"Setting PL/I Compiler Options - Hardcoding" on page 60.

"Setting PL/I Compiler Options - From Builder Definition" on page 61.

"Setting PL/I Compiler Options - Based on Work Area Name" on page 62.

"Reducing the Number of Build Messages" on page 64.

Although "Setting PL/I Compiler Options - Hardcoding" on page 60, "Setting PL/I Compiler Options - From Builder Definition" on page 61, and "Setting PL/I Compiler Options - Based on Work Area Name" on page 62, describe handling parameters for the PL/I compiler, the same techniques can also be applied to the DB2 precompiler and CICS translator options.

References to comments in the following sections refer to the comment numbers in Figure 6 on page 9 and Figure 8 on page 11.

KEEPING MVSPLI.JCL AT THE MVS HOST

You might want to keep mvspli.jcl (Figure 6 on page 9) on the MVS Host system, rather than maintaining it in TeamConnection. If several builders all share the same build script (mvspli.jcl), the advantage of maintaining the build script on the MVS host is that you do not need to update each builder to pick up the revised build script. The disadvantage is that when mvspli.jcl is changed, TeamConnection is not aware of the change and does not automatically rebuild all programs whose builder has been changed due to the change in the build script. This could be an advantage if you frequently change mvspli.jcl to add new functions that do not require existing modules to be rebuilt.

To use an MVS host copy of mvspli.jcl, make the following changes:

Make sure that the current copy of mvspli.jcl is in the data set pointed to by the TEAMPROC DD statement in RUNPGMT.

© Copyright IBM Corp. 1997

Modify the properties for the 4 builders (mvspli, mvsdb2pli, mvscicspli, and mvsdb2cicspli) that use mvspli.jcl as the build script. When you modify the builders' properties, set the **File type** to **None**.

KEEPING MVSRXPLI.RXP AT THE MVS HOST

You might want to keep mvsrxpli.rxp (Figure 8 on page 11) on the MVS Host system, rather than maintaining it in TeamConnection. The advantage of storing mvsrxpli.rxp on the MVS host is that it simplifies the build tree. The disadvantage is that when mvsrxpli.rxp is changed, TeamConnection is not aware of the change and does not automatically rebuild all programs that use mvsrxpli.rxp as part of the build. This could be an advantage if you frequently change mvsrxpli.rxp to add new functions that do not require existing modules to be rebuilt.

To use an MVS host copy of mvsrxpli.rxp, make the following changes:

In mvspli.jcl, change the SYSEXEC DD statement to point to the data set where MVSRXPLI resides. For example, the new DD statement would be something like:

```
//SYSEXEC   DD  DSN=JERI.TEAMC.REXXCMDS,DISP=SHR
```

If you have not stored mvspli.jcl at the MVS host, then to put this change into effect, you must modify the properties for the 4 builders (mvspli, mvsdb2pli, mvscicspli, and mvsdb2cicspli) that use mvspli.jcl as the build script. When you modify the builders' properties, set the **Source file** to point to the path and directory for the updated mvspli.jcl.

Upload mvsrxpli.rxp to the MVS host and store it as member MVSRXPLI in the data set referenced by SYSEXEC (JERI.TEAMC.REXXCMDS in this example).

Remove mvsrxpli.rxp from the build trees for all PL/I source programs.

If you use crtplibt.cmd as shown in Appendix B, “SAMPLE REXX TO CREATE BUILD TREES” on page 91, modify it so that it does not connect mvsrxpli.rxp to the .obj part.

Make any future changes to the copy of mvsrxpli.rxp that is stored at the MVS host.

SUPPORTING PROGRAM PRODUCT INCLUDE LIBRARIES

If all include members are provided by a program product and are maintained only on the MVS host, not within TeamConnection, make the following modifications:

In mvspli.jcl (Figure 6 on page 9), change the INCSRC DD statement to point to the program product library that contains the include members. For example, the new DD statement should be something like:

```
//INCSRC   DD  DSN=program.product.include.library,DISP=SHR
```

If you have not stored mvspli.jcl at the MVS host, then to put this change into effect, you must modify the properties for the 4 builders (mvspli, mvsdb2pli, mvscicspli, and

mvsdb2cicspli) that use mvspli.jcl as the build script. When you modify the builders' properties, set the **Source file** to point to the path and directory for the updated mvspli.jcl.

Do not specify a parser for the PL/I source programs. This is because you do not want TeamConnection to detect dependencies for any of the include members.

You can remove the parser from a part by modifying the part's properties and setting the **Parser** to **NULL**.

SUPPORTING PROGRAM PRODUCT INCLUDE LIBRARIES CONCATENATED WITH TEAMCONNECTION COPYBOOKS

If some include members are provided by a program product and are maintained only on the MVS host and other include members are developed locally and maintained within TeamConnection, make the following modifications:

In mvspli.jcl (Figure 6 on page 9), change the INCSRC statement to point to a permanent data set on the MVS host that contains a copy of the include members that are maintained within TeamConnection. For example, the new DD statement should be something like:

```
//INCSRC     DD  TCEXT=INC,DSN=JERI.TEAMC.MYINC.&TCRELEAS.,DISP=OLD
```

Using TCEXT=INC causes TeamConnection to place any include members identified as dependencies of the PL/I source program in the data set specified by the DSN parameter. If the include member already exists in the data set, the include member is replaced.

Using &TCRELEAS as part of the data set name allows you to maintain separate include libraries on the host, based on the name of the release within TeamConnection.

If you have not stored mvspli.jcl at the MVS host, then to put this change into effect, you must modify the properties for the 4 builders (mvspli, mvsdb2pli, mvscicspli, and mvsdb2cicspli) that use mvspli.jcl as the build script. When you modify the builders' properties, set the **Source file** to point to the path and directory for the updated mvspli.jcl.

In mvsrxpli.rxp (Figure 8 on page 11), make the following changes:

- In the "INITIALIZATION" section near comment (1), add a statement to initialize a variable to the name of the program product library that contains the include members. For example, add:

  ```
  hostonlylib = 'program.product.include.library'

  where program.product.include.library is the actual name
        of the library on the MVS host, for example: CICS33 .SDHFPL1
  ```

- In the precompile routine at comment (6), change the allocation for SYSLIB to:

  ```
  "ALLOC FI(SYSLIB) DA('"hostonlylib"','"cpysrc"') SHR REUSE"
  ```

The ALLOC command requires all data sets in the concatenation to be catalogued. Specifying a permanent (catalogued) data set for the INCSRC DD statement in mvspli.jcl makes this possible.

– In the compile routine at comment (17), change the allocation for SYSLIB to:

```
"ALLOC FI(SYSLIB) DA('"hostonlylib"','"cpysrc"') SHR REUSE"
```

Specify a parser for the PL/I source program and any include members that are stored in TeamConnection. The PL/I parser (fhbplprs.cmd) provided with TeamConnection, identifies only the include members that are stored in TeamConnection as dependencies of the PL/I source program. Include members that are not stored in TeamConnection are ignored by the parser and are not added as dependencies of the PL/I source program. This provides support for a mixture of include members stored in TeamConnection and in an MVS program product library.

Note: If you use this technique, when you **View Build Messages**, you might see an error message like the following:

```
6 21-647  Data set incompatibility, particularly the lrecl, will prevent
            any copying from 'DD:TC$INC(PLIPGM)' TO 'DD:TC    17'
```

This message typically appears when the program being built uses an existing include member that has not been changed.

Verify that:

New include members are copied to the MVS host

Changes to include members are copied to the MVS host

If the above are true, then as long as the object modules are built correctly, you can ignore this message.

SETTING PL/I COMPILER OPTIONS - HARDCODING

Figure 8 on page 11 shows one technique for setting compiler options -- hard coding them in mvsrxpli.rxp. For example, at comment (19), the choice of compiler options is made based on whether the PL/I source program uses CICS commands or not.

```
pliopts = 'INCLUDE,MACRO,OBJECT,NODECK,MAP,OPT(2),NOSOURCE'
if cics = 'Y' then
   pliopts = pliopts || ',SYSTEM(CICS)'

"CALL '"plicomp"(IEL AA)' '"pliopts""'
```

The advantage of this technique is that it is easy to understand. The disadvantage is that all the PL/I source programs that use the builder must specify the same compiler options. This can be an advantage in enforcing your installation standards.

SETTING PL/I COMPILER OPTIONS - FROM BUILDER DEFINITION

Another technique for setting PL/I compiler options is to specify them in the **Parameters** field when you define a builder. To specify compiler options on the builder, make the following changes:

In the definition of the builder(s), change the parameters to include PLIOPT(options). For example, for the mvspli builder, the parameters change

From
'MVSRXPLI N N Y'

To
'MVSRXPLI N N Y PLIOPT(INCLUDE,OPT(2),SOURCE)'

In mvsrxpli.rxp (Figure 8 on page 11), make the following changes:

– Change the parse statement at comment (1) to:

```
parse arg db2 cics plicmp pliopt otherparms
```

– After setting the required PL/I compiler options at comment (19), add logic to parse out the PL/I compiler options before calling the PL/I compiler. For example, change:

From:

```
pliopts = 'INCLUDE,MACRO,OBJECT,NODECK,MAP,OPT(2),NOSOURCE'
if cics = 'Y' then
   pliopts = pliopts || ',SYSTEM(CICS)'

"CALL '"plicomp"(IEL AA)' '"pliopts"'"
```

To:

```
pliopts = 'INCLUDE,MACRO,OBJECT,NODECK,MAP,OPT(2),NOSOURCE'
if cics = 'Y' then
   pliopts = pliopts || ',SYSTEM(CICS)'

/  If PL/I options supplied to builder, let them replace defaults       /
if substr(pliopt,1,7) = 'PLIOPT(' then
  do
     plioptl = length(pliopt) - 8              /  subtract pre/suffix          /
     if plioptl >   then
        pliopts = substr(pliopt,8,plioptl)
  end

"CALL '"plicomp"(IEL AA)' '"pliopts"'"
```

The advantage of this technique is that it is clear from the builder definition which compiler options are being used.

The disadvantage of this technique is that there is a fixed (about 100 bytes) amount of space allotted for the entire parameter string for a builder. If the technique is extended to support both DB2 precompiler and CICS translator options, the parameter string for the builder might be too long.

You could modify this technique to allow the builder to specify some options, while other options are hard-coded.

SETTING PL/I COMPILER OPTIONS - BASED ON WORK AREA NAME

You might need to vary the PL/I compiler options based on the work area and/or driver that is currently being built. For example, during unit and system test you might use the NOOPTIMIZE compiler option and during acceptance test you might use the OPTIMIZE(TIME) compiler option.

To vary the compiler options based on the work area or driver that is being built and assuming that you did not make the changes described in “Setting PL/I Compiler Options - From Builder Definition” on page 61, make the following changes:

Change the parameters for the 4 builders (mvspli, mvsdb2pli, mvscicspli, and mvsdb2cicspli) that use mvspli.jcl as the build script. For example, for mvspli, change the parameters:

From
```
'MVSRXPLI N N Y'
```

To
```
'MVSRXPLI N N Y &TCWKAREA'
```

This includes the work area or driver name as part of the parameter list.

In mvsrxpli.rxp (Figure 8 on page 11), do the following:

- Change the parse statement at comment (1) to include the work area name.

  From
  ```
  parse arg db2 cics plicmp otherparms
  ```

  To
  ```
  parse arg db2 cics plicmp wkareanam otherparms
  ```

- After setting the required PL/I compiler options at comment (19), add logic to modify the PL/I compiler options based on the work area name. For example, change:

From:

```
pliopts = 'INCLUDE,MACRO,OBJECT,NODECK,MAP,OPT(2),NOSOURCE'
if cics = 'Y' then
   pliopts = pliopts || ',SYSTEM(CICS)'

"CALL '"plicomp"(IEL AA)' '"pliopts"'"
```

To:

```
pliopts = 'INCLUDE,MACRO,OBJECT,NODECK,MAP,NOSOURCE'
if cics = 'Y' then
   pliopts = pliopts || ',SYSTEM(CICS)'

/  Modify PL/I options based on work area name                    /
if wkareanam = 'INITWA' then
      pliopts = pliopts || ',NOOPTIMIZE'
else
      pliopts = pliopts || ',OPTIMIZE(TIME)'

"CALL '"plicomp"(IEL AA)' '"pliopts"'"
```

The work area name is translated to upper case when it is placed in the parameter list. This shows combining the use of the &TCWKAREA value in wkareanam with the cics parameter that was explicitly set in the builder to determine which PL/I compiler options should be used.

&TCRELEAS can also be used in the builder parameter list. &TCRELEAS provides the name of the release for which the build is being done.

The advantage of this technique is that it allows you to vary the compiler options based on the work area, driver, and/or release that is being built.

The same technique can be used to set the names of the DB2 precompiler, CICS translator, or PL/I compiler libraries or anything else that might vary by work area, driver, and/or release.

The disadvantage of this technique is that you must adhere to some naming convention for work areas and drivers so that mvsrxpli.rxp can distinguish between work areas and different drivers in order to set the PL/I compiler options. You must also establish naming conventions for releases if you need to vary options based on the release name.

REDUCING THE NUMBER OF BUILD MESSAGES

You might want to change the output messages that are passed back to TeamConnection. You can minimize the output messages by varying the DB2 precompiler, CICS translator, and PL/I compiler options. For example, specifying the NOSOURCE option for the CICS translator reduces the amount of information returned in the build messages.

Alternatively, you might decide that you only want to see error messages. errorhndlr is used when an error occurs to copy the information for the failing step to the build messages (the ERRLIST DD statement in mvspli.jcl). To modify mvsrxpli.rxp so that the build messages are minimized and deal primarily with errors, replace the calls to TCOUThndlr with statements that free the output files. The following changes are needed:

In the precompile step at comment (10), change:

From:

```
call TCOUThndlr PRECOMPILE,precompile_rc,SYSTERM
```

To:

```
"FREE  FI(SYSTERM)"
```

In the translate step at comment (15), change:

From:

```
call TCOUThndlr TRANSLATE,translate_rc,SYSPRINT
```

To:

```
"FREE  FI(SYSPRINT)"
```

In the compile step at comment (21), change:

From:

```
call TCOUThndlr COMPILE,compile_rc,SYSPRINT
```

To:

```
"FREE  FI(SYSPRINT)"
```

HINTS AND TIPS FOR MVS BUILDS

The following provide some debugging hints and tips for MVS build processors.

Need to trace what is happening

Do one or both of the following:

Add the -T parameter to the CALL statement in the RUNPGMT job. The -T provides a trace of the TeamConnection commands that are running.

Add

```
TRACE INTERMEDIATES
```

as the second line of mvsrxpli.rxp. This turns on tracing for the REXX command file.

Build agent fails

One cause for this is an error in the REXX command file, possibly due to:

Mismatched quotation marks.

Line length too long.

Either of these can occur from a change you just made. Cancel the MVS build processor and examine the trace output to determine the cause of the problem.

Build fails even after you have "fixed" the build script.

Check that you modified the properties for the build script and reentered the path and file name for the script. This is required to make TeamConnection pick up a fresh copy of the build script.

You cancelled a build, and try to start another build for the same part, but get a message that there is already a build for the same part on the job queue.

Change the **Timeout** value for the builder to 1. Start the build agent for the pool for which the build was done. The build should now time out and remove the part from the job queue.

Remember to change the **Timeout** value back to the original value before starting the build again.

© Copyright IBM Corp. 1997

You need 2 sets of in-stream data, possibly one with just &TCINPUT and another with just &TCOUTPUT.

Assuming that the 2 sets of in-stream data are at the end of the build script, use the following statements:

```
 //INLIST DD
&TCINPUT
/
 //OUTLIST DD
&TCOUTPUT
//
```

This results in both &TCINPUT and &TCOUTPUT being available. However, the following also occur and must be handled in the REXX command file:

/* appears as the last line of input for the INLIST DD statement.

A blank line appears as the first line of input for the OUTLIST DD statement due to the /* to end the INLIST DD statement.

The following 2 techniques do **NOT** work:

Technique 1 - fails because the first // ends the build script and the OUTLIST DD statement cannot be found by the REXX command file

```
 //INLIST DD
&TCINPUT
//
 //OUTLIST DD
&TCOUTPUT
//
```

Technique 2 - fails because MSGLIST can not be found; processing is not transferred to the REXX command file

```
 //INLIST DD
&TCINPUT
 //MSGLIST DD TCEXT=MSG, ....
 //OUTLIST DD
&TCOUTPUT
//
```

Need to run 2 MVS build processors

Be sure to:

Start 2 different build agents with different socket numbers.

Use different job names for the MVS build processors, with each job specifying a different socket number that corresponds to the socket number specified by one of the build agents.

No data sets in the jobs have DISP=OLD

The job class being used supports sufficient initiators. Each build processor requires its own initiator.

Able to ping the MVS machine, but get a TCP/IP error trying to run either RUNPGM or RUNPGMT.

If the error message is similar to:

```
RC=1 11 on IUCV, IUCV CONNECT TO TCP/IP, FD=-254, PATH= ,
  IPR CODE=11, LUCV NAME=  de9a
```

and in TeamConnection, you get something similar to:

```
1 - 63  error edc8 4i, request failed because of IUCV error
            when the TeamConnection server attempted to process
            the socket function
```

check whether the TCP/IP naming conventions for your installation have been changed from the default values. If so, you might need a SYSTCPD DD statement to point to the data set that defines the different naming conventions.

APPENDIX A. BUILDER, BUILD SCRIPT, AND REXX FOR LINK EDIT AND BIND

This section describes how the builder definition, the build script, and the REXX command file work together to control the build for the load module. This build involves linking the object modules together as well as doing the binds for any DBRMs that correspond to the object modules.

DEFINING THE BUILDER

When IRXJCL is used in the build script, the parameters for the builder define which REXX command file is used during the build. For example, in the builder definition:

```
Builder         mvsplilnk
Release         release1
Script          mvslnk.jcl
Environment     mvstso
Comp. oper.     ==
RC value
File type       none
Source file
 Parameters     'MVSRPLNK N N N NA NA &TCWKAREA'
Timeout         4
```

the parameters are:

MVSRPLNK Name of the REXX command file that is run by IRXJCL. IRXJCL always expects the first parameter to be the name of the REXX command file.

N N N NA NA &TCWKAREA Correspond to the parameters expected by MVSRPLNK.

This builder assumes that the build script, shown in Figure 28 on page 70, is stored on the MVS host. The TEAMPROC DD statement in RUNPGMT points to the partitioned data set in which mvslnk.jcl is stored.

© Copyright IBM Corp. 1997

NOTES ON MVSLNK.JCL (THE BUILD SCRIPT)

The following figure shows the build script that supports link edit and bind.

```
//REXX       EXEC   PGM=IRXJCL,
// PARM=&TCPARM
//
//  BUILD SCRIPT FOR EXECUTING A REXX PROGRAM
//
//  TEAMPROC MUST RUN IN A TSO ADDRESS SPACE
//
//  IRXJCL RUNS THE PROGRAM NAMED IN THE FIRST PARAMETER OF &TCPARM
//  AS A REXX PROGRAM AND PASSES THE REMAINING PARAMETERS
//
//  IRXJCL LOOKS INTO SYSEXEC FOR THE REXX PROGRAM
//
//SYSEXEC   DD  DSN=JERI.TEAMC.REXXCMDS,DISP=SHR                    (1)
//DBRM       DD  TCEXT=DBR,DISP=(NEW,DELETE),
//             UNIT=SYSDA,SPACE=(32   ,(3 ,1 ,5)),
//             DCB=(RECFM=FB,LRECL=8 ,BLKSIZE=616 )
//OBJECT    DD  TCEXT=OBJ,DISP=(NEW,DELETE),
//             UNIT=SYSDA,SPACE=(32   ,(3 ,1 ,5)),
//             DCB=(RECFM=FB,LRECL=8 ,BLKSIZE=32  )
//LINKCTL   DD  TCEXT=LNK,DISP=(NEW,DELETE),                        (2)
//             UNIT=SYSDA,SPACE=(32   ,(3 ,1 )),
//             DCB=(RECFM=FB,LRECL=8 ,BLKSIZE=32  )
//SYSLMOD    DD TCOUT=LKD,DSN=JERI.TEAMC.LOAD.&TCRELEAS.,DISP=SHR (3)
//ERRLIST   TCEXT=TCOUT,DISP=(NEW,DELETE),
//             UNIT=SYSDA,SPACE=(133  ,(3 ,3 )),
//             DCB=(RECFM=FB,LRECL=133,BLKSIZE=133 )
//MSGLIST   TCEXT=MSG,DISP=(NEW,DELETE),
//             UNIT=SYSDA,SPACE=(133  ,(3 ,3 )),
//             DCB=(RECFM=FB,LRECL=121,BLKSIZE=121 )
//INLIST     DD                                                     (4)
&TCINPUT
/
//OUTLIST   DD                                                      (5)
&TCOUTPUT
//
```

Figure 28. mvslnk.jcl - Build Script for Link Edit and Bind

The following notes correspond to the numbers in the right margin of the mvslnk.jcl build script, shown in Figure 28 on page 70. The notes explain concepts used in the build script.

Note:

1. The SYSEXEC DD statement must point to the library where the REXX command file resides. In this example, the REXX command file (mvslnk.jcl) must be stored in JERI.TEAMC.REXXCMDS on the MVS host.
2. The LINKCTL DD statement is the temporary library where the link edit statements are stored. Because there should only be one link edit control statement for a link edit, this data set is allocated as a sequential file. This avoids having to determine the member name to specify in the command that allocates the SYSLIN file for the linkage editor.
3. The SYSLMOD DD statement is where the final load module is stored on the MVS host. &TCRELEAS is used to show how the data set name can be varied based on one of the TeamConnection parameters. &TCWKAREA is another TeamConnection parameter that can used in the build script as part of a data set name.

   Because the file type for the .lkd file is **None** when the part is defined to TeamConnection, the load module is not transferred to TeamConnection for storage. This avoids transferring large load modules back to TeamConnection.
4. The INLIST DD statement is used by mvsrplnk.rxp to obtain the names of the input parts. &TCINPUT results in multiple lines of in-stream data, with each line corresponding to one input part. The input parts included are based on the build tree. The file extensions are not included.

   Using a naming convention where the object module name and DBRM member have the same part name except for the file extension allows mvsrplnk.rxp to determine the name(s) of the DBRM member(s) that must be bound. For example, if the object module name is plidb2.obj, the corresponding DBRM name is plidb2.dbr.

   Because there are 2 DD statements for in-stream data (INLIST and OUTLIST), the INLIST DD statement is ended by the /*. Unlike normal in-stream data, the /* is included as the last line of input data.
5. The OUTLIST DD statement is used by mvsrplnk.rxp to obtain the names of the output parts. The load module (.lkd) is created with a file type of **None**. This avoids the overhead of bringing down the load module to the TeamConnection family. However, it also means that the .lkd part is not included with &TCOUTPUT.

   The build trees for plidb2 and plidcic include a .msg part as an output from the build of the load module. See Figure 16 on page 27 for an example. Use a naming convention that the load module (.lkd), link edit control statements (.lnk), and message (.msg) parts all have the same part name except for the file extension.

Unlike normal in-stream data, the /* that ends the INLIST DD statement causes a blank line to be included at the beginning of the OUTLIST input, before the first of the output parts is listed.

The algorithm used by mvsrplnk.rxp to determine which parts must be bound is the following:

Using INLIST, build 2 arrays based on the input:

- unique part name
- number of times the unique part name appears in INLIST

Every .obj / .dbr pair counts 2.

If the .lnk part is named the same as the .obj / .dbr pair, then the count is 3.

Exclude the /* from the list of unique part names

Using OUTLIST, find the part name for the .msg part. There should only be one part name listed within OUTLIST. Allow for any blank lines at the beginning of the OUTLIST in-stream data.

Find the part name from OUTLIST in the array of unique names that was built from INLIST. Subtract 1 from the number of occurrences for this unique name.

For each of the unique names in the array, if the number of occurrences is 2 or more, then the name represents a .obj / .dbr pair and a BIND needs to be done for the DBRM. A .lnk / .msg pair might have also existed with the same name, but the pair has been eliminated from the count.

If the number of occurrences is 0, this represents a .lnk / .msg pair, where the .msg part balanced out the .lnk part.

If the number of occurrences is 1, this represents a situation where the .obj, .lnk, and .msg parts all have the same part name, but there is no DBRM. After the .lnk / .msg pair is eliminated from the count, only the .obj part remains.

NOTES ON MVSRPLNK.RXP (THE REXX COMMAND FILE)

The following figure shows the REXX command file that supports link edit and bind.

```
/ REXX /
TRACE INTERMEDIATES

parse arg db2 dli cics db2id db2col wkareanam otherparms          (1)
/                                                  /
/   INITIALIZATION                                                          /
/                                                  /
                                                                          (2)
wspc          = 5                          / Space allocation default     /

dsnload       = 'DSNA.DSNLOAD'             / DB2 libraries                /
dfhload       = 'CICS33 .SDFHLOAD'         / CICS libraries               /

plilib        = 'SYS1.PLI.PLIBASE'         / PL/I libraries               /
plilib2       = 'SYS1.PLI.SIBMBASE'
plilib3       = 'SYS1.PLI.PLITASK'
plilib4       = 'SYS1.PLI.SIBMLINK'

reslib        = 'SYS1.RESLIB'              / IMS libraries                /
linklib       = 'SYS1.LINKLIB'             / Linkage editor library       /

say 'wkareanam =' wkareanam
say 'otherparms=' otherparms

/                                                  /
/ RETRIEVE DATASET NAMES FOR FILES DEFINED IN THE BUILD SCRIPT            /
/                                                  /
                                                                          (3)
dsinfo = getdsn_last('DBRM')
if dsinfo >   then
   call errorhndlr GETDBRM,dsinfo
dbrmlib = SYSDSNAME

dsinfo = getdsn_last('OBJECT')
if dsinfo >   then
   call errorhndlr GETOBJECT,dsinfo
objlib = SYSDSNAME

dsinfo = getdsn_last('LINKCTL')
if dsinfo >   then
   call errorhndlr GETLINKCTL,dsinfo
linkctl = SYSDSNAME
```

Figure 29 (Part 1 of 9). mvsrplnk.rxp - REXX Command File Supporting Link Edit and Bind

```
dsinfo = getdsn_last('SYSLMOD')
if dsinfo >   then
   call errorhndlr GETSYSLMOD,dsinfo
syslmod = SYSDSNAME

/                                                  /
/  FORCE DB2, DLI, AND CICS PARAMETERS TO Y OR N                                  /
/                                                  /
                                                                                  (4)
if db2          <> 'Y' then
  db2               =  'N'
if dli          <> 'Y' then
  dli               =  'N'
if cics         <> 'Y' then
  cics              =  'N'

/                                                  /
/  SET UP PL/I OPTIONS                                                          /
/                                                  /
                                                                                  (5)
/  if wkareanam = 'INITWA' then
        < set options >
else
        < set options >                                  /

/                                                  /
/  CALL APPROPRIATE ROUTINES                                                      /
/                                                  /
                                                                                  (6)
call link
if db2 = 'Y' then
  do
     call  getdbrmnams
      firstbind = 'y'
      do i = 1 to dbrmmbr.
         say dbrmmbr.i dbrmcnt.i                 /  SAY used for debugging          /
         if dbrmcnt.i >= 2 then                  /  Need OBJ/DBR pair                /
           do
              call  bind
               firstbind = 'n'
           end
     end
      if firstbind = 'n' then                  /  Close .msg file                  /
         "execio   diskw MSGLISTX (finis)"
  end

exit
```

Figure 29 (Part 2 of 9). mvsrplnk.rxp - REXX Command File Supporting Link Edit and Bind

```
link:
/                                                  /
/  LINK THE OBJECT MODULES                                                    /
/                                                  /
                                                                          (7)
/  Order of link libraries varies based on whether CICS is involved     /
if cics = 'Y' then
  do
     link_lib = """dfhload""" """plilib""" ,
                  """plilib2""" """plilib3""" """plilib4"""
     if db2 = 'Y' then
        link_lib = link_lib """dsnload"""
     if dli = 'Y' then
        link_lib = link_lib """reslib"""
  end
else
  do
     link_lib = """plilib""" """plilib2""" """plilib3""" """plilib4"""
     if dli = 'Y' then
        link_lib = link_lib """reslib"""
     if db2 = 'Y' then
        link_lib = link_lib """dsnload"""
  end

/  Optionally allocate files for program product libraries here          /
                                                                          (8)
"ALLOC FI(SYSLIB)      DA("link_lib") SHR REUSE"
"ALLOC FI(SYSLIN)      DA("'linkctl'") SHR REUSE"
"ALLOC FI(OBJLIB)      DA("'objlib'") SHR REUSE"
"ALLOC FI(SYSLMOD)    DA("'syslmod'") SHR REUSE"
/  block sized changed from 8   to 121     for SYSPRINT     /
"ALLOC FI(SYSPRINT) BLOCK(121 ) SPACE("wspc","wspc") UNIT(SYSDA) REUSE"
"ALLOC FI(SYSUT1)      BLOCK(1 24) SPACE("wspc","wspc") UNIT(VIO) REUSE"
                                                                          (9)
"CALL '"linklib"(IEWL)' 'RENT,REUS,LET,LIST,XREF,MAP,",
         "AMODE(31),RMODE(ANY)'"
link_rc = rc

"FREE FI(SYSLIB,SYSLIN,OBJLIB,SYSLMOD,SYSUT1)"
if link_rc <  | link_rc > 4 then                                    (1 )
   call errorhndlr LINK,link_rc,SYSPRINT
else
   call TCOUThndlr LINK,link_rc,SYSPRINT                           (11)

return
```

Figure 29 (Part 3 of 9). mvsrplnk.rxp - REXX Command File Supporting Link Edit and Bind

```
bind:
/                                          /
/  BIND EXECUTES A DB2 BIND                                                  /
/                                          /
                                                                        (12)
"ALLOC FI(DBRMLIB)    DA('"dbrmlib"') SHR REUSE"
"ALLOC FI(SYSPRINT) BLOCK(121 ) SPACE("wspc","wspc") UNIT(SYSDA) REUSE"
"ALLOC FI(BINDIN)     BLOCK(8  ) SPACE("wspc","wspc") UNIT(SYSDA) REUSE"
                                                                        (13)
drop bindctl bindbctl bindactl
                                                                        (14)
/  The following assumes that packages are used and that the bind        /
/  control statements can be in the REXX command file and controlled  /
/  with parameters to the builder                                              /
bindctl.1    = 'DSN SYSTEM('db2id')'
bindctl.2    = 'BIND PACKAGE('db2col') -'
bindctl.3   =  '          MEMBER('dbrmmbr.i')  -'
bindctl.4   =  '          ACT(REP)  -'
bindctl.5   =  '          VALIDATE(BIND)  -'
bindctl.6   =  '          ISOLATION(CS)'
bindctl.7    = '         ENABLE(xxx)'              /  Optional, if use fix           /
bindctl.8    = '         OWNER(xxxxxxx)'          /  comment & continuation mark /

/  CLIST statements that must be included before the BIND command        /
bindbctl.1 = 'SET BINDRC =  '
bindbctl.2 = 'ERROR DO'
bindbctl.3 = '    SET BINDRC = &LASTCC'
bindbctl.4 = '    IF &SYSPCMD = &SYSSCMD THEN GOTO BINDEXIT'
bindbctl.5 = '    ELSE RETURN'
bindbctl.6 = 'END'

/  CLIST statements that must be included after the BIND command          /
bindactl.1 = 'END'
bindactl.2 = 'BINDEXIT: ERROR OFF'
bindactl.3 = 'EXIT CODE(&BINDRC)'
```

Figure 29 (Part 4 of 9). mvsrplnk.rxp - REXX Command File Supporting Link Edit and Bind

```
                                                                    (15)
/  Build a CLIST that can run the BIND commands                      /
"EXECIO   DISKW BINDIN ( STEM bindbctl. )"
"EXECIO   DISKW BINDIN ( STEM bindctl. )"
"EXECIO   DISKW BINDIN ( STEM bindactl. OPEN FINIS )"
dsinfo = getdsn_last('BINDIN')
bindin_dsn = SYSDSNAME
                                                                    (16)
/  Trap all output so that it goes to the bindout file            /
drop bindout
bindtrap = outtrap("bindout.")
                                                                    (17)
/  Run the CLIST that was just created                            /
"EX '"bindin_dsn"'"
bind_rc = rc
bindtrap = outtrap(OFF)
"execio   diskw SYSPRINT (stem bindout. open finis)"

if firstbind = 'y' then                                             (18)
  do
     dsinfo = getdsn_last('MSGLIST')
     if dsinfo >   then                 /  Save BIND messages in MSGLIST DD /
        call errorhndlr GETMSGLIST,dsinfo
     msglist = SYSDSNAME
     "ALLOC FI(MSGLISTX)      DA('"msglist"') SHR REUSE"
     "execio   diskw MSGLISTX (open)"
  end
"execio   diskw MSGLISTX (stem bindout.)"

"FREE FI(BINDIN DBRMLIB)"
if bind_rc <   | bind_rc > 4 then                                   (19)
   call errorhndlr BIND,bind_rc,SYSPRINT
else
  "FREE  FI(SYSPRINT)"                                              (2 )

return
```

Figure 29 (Part 5 of 9). mvsrplnk.rxp - REXX Command File Supporting Link Edit and Bind

```
getdsn_last: procedure expose sysdsname sysmsglvl1 sysmsglvl2
/                                                   /
/    DETERMINE DATA SET NAME OF A TEMPORARY FILE                                  /
/                                                   /
parse arg ddname parm_rest
getdsn ddname
getdsn_rc = rc
if getdsn_rc <   | getdsn_rc >   then
   sysdsname = ''
else
  do
     last_dsn = sysdsname.
     sysdsname = sysdsname.last_dsn
  end
return getdsn_rc


getdbrmnams: procedure expose dbrmmbr. dbrmcnt.                    (21)
/                                                   /
/    DETERMINE NAME OF OBJECT/DBRM MODULES FOR USE AS DBRM MEMBER NAME /
/                                                   /
dsinfo = getdsn_last('INLIST')              /  List of .OBJ, .DBR, .LNK      /
if dsinfo >   then
   call errorhndlr GETINLIST,dsinfo
inlist = SYSDSNAME

dsinfo = getdsn_last('OUTLIST')             /  List of .LKD (should be 1)  /
if dsinfo >   then
   call errorhndlr GETOUTLIST,dsinfo
outlist = SYSDSNAME

myinput.    = ''                               /  Clear arrays                  /
myoutput. = ''
dbrmmbr.   =  ''
dbrmmbr.  =
dbrmcnt.   =

"ALLOC FI(INLISTX) DA('"inlist"') SHR REUSE"
"execio   diskr INLISTX (stem myinput. open finis)"
```

Figure 29 (Part 6 of 9). mvsrplnk.rxp - REXX Command File Supporting Link Edit and Bind

```
/  INLIST contains one line for each input in the build tree.              /
/  Because the file extension is removed, the OBJ and DBRM outputs       /
/  and possibly the link edit command file cannot be distinguished.     /
/  Using the naming convention of xxxxx.OBJ and xxxxx.DBR for the        /
/  object module and corresponding DBRM allows you to determine the     /
/  OBJ/DBR pairs.    Using the naming convention that the xxxxx.LNK       /
/  xxxxx.LKD provide the link edit control statements and                 /
/  corresponding load module means that you can determine which of       /
/  the lines in INLIST corresponds to the LNK part by looking at the  /
/  OUTLIST DD statement.    OUTLIST should only have one line in it --  /
/  the line for the load module that is being built.                      /

do i = 1 to myinput.
   say myinput.i                                  /  SAY used for debugging          /

   if word(myinput.i,1) <> '/ ' then        /  skip eof marker                    /
    do
      seenit = 'n'                                  /  check if saw name before  /
      jmax = dbrmmbr.
      do j = 1 to jmax while seenit = 'n'
         say seenit                                    /  SAY used for debugging       /
         say j dbrmmbr.j dbrmcnt.j                  /  SAY used for debugging       /
         if word(myinput.i,1) = dbrmmbr.j then /  Not new name                  /
           do
              dbrmcnt.j = dbrmcnt.j + 1           /  Bump count for this name  /
              seenit = 'y'
           end
     end
      if seenit = 'n' then                          /  Not yet in dbrmmbr array  /
        do
            say j                                       /  SAY used for debugging       /
            dbrmmbr.  = dbrmmbr.  + 1             /  Bump unique names count     /
            dbrmmbr.j = word(myinput.i,1)
            dbrmcnt.j = 1                               /  Init count for this name  /
        end
    end                                                 /  end check on eof mark        /
end

/  Get name of the load module from the OUTLIST DD statement                /
/  Do not count the name of the load module in determining if there     /
/  is an OBJ/DBR pair of inputs                                                    /
"ALLOC FI(OUTLISTX) DA('"outlist"') SHR REUSE"
"execio   diskr OUTLISTX (stem myoutput. open finis)"
```

Figure 29 (Part 7 of 9). mvsrplnk.rxp - REXX Command File Supporting Link Edit and Bind

```
seenit = 'n'
do i = 1 to myoutput.  while seenit = 'n' /  Loop til non-blank line  /
   loadmod = word(myoutput.i,1)              /  Get name of load module        /
   say myoutput.i                             /  SAYs used for debugging        /
  say  loadmod
   if loadmod <> "" then
      seenit = 'y'
end

seenit = 'n'
do j = 1 to dbrmmbr.  while seenit = 'n' /  Go through possible DBRMs /
   if dbrmmbr.j = loadmod then
     do
        dbrmcnt.j = dbrmcnt.j - 1
        seenit = 'y'
     end
end

return


TCOUThndlr: procedure expose sysmsglvl1 sysmsglvl2
/                                                  /
/    ISSUE ERROR MESSAGE FOR FUNCTION, ERROR CODE, AND DATA SET            /
/                                                  /
parse arg function, return_code, copyddname
dsinfo = getdsn_last('ERRLIST')
"ALLOC FI(ERROUT) DA('"SYSDSNAME"') MOD REUSE"
if copyddname <> " then
  do
     "execio   diskr "copyddname" (stem errvar. open finis)"
     "execio   diskw errout (STEM errvar. open finis)"
    "FREE  FI("copyddname")"
  end
else
   'EXECIO 2 DISKW ERROUT (STEM sysmsglvl OPEN FINIS)'
"FREE FI(ERROUT)"
return
```

Figure 29 (Part 8 of 9). mvsrplnk.rxp - REXX Command File Supporting Link Edit and Bind

```
errorhndlr: procedure expose sysmsglvl1 sysmsglvl2
/                                                   /
/    ISSUE ERROR MESSAGE FOR FUNCTION, ERROR CODE, AND DATA SET          /
/                                                   /
parse arg function, return_code, copyddname
dsinfo = getdsn_last('ERRLIST')
"ALLOC FI(ERROUT) DA('"SYSDSNAME"') MOD REUSE"
if copyddname <> '' then
  do
      "execio   diskr "copyddname" (stem errvar. open finis)"
      "execio   diskw errout (STEM errvar. open finis)"
     "FREE  FI("copyddname")"
  end
else
   'EXECIO 2 DISKW ERROUT (STEM sysmsglvl OPEN FINIS)'
"FREE FI(ERROUT)"
exit(return_code)
```

Figure 29 (Part 9 of 9). mvsrplnk.rxp - REXX Command File Supporting Link Edit and Bind

The following notes correspond to the numbers in the right margin of the mvsrplnk.rxp REXX command file, shown in Figure 29 on page 74. The notes explain concepts used in the REXX command file.

Note:

1. The parameters for mvsrxpli.rxp are:

   db2 Y/N flag indicating whether DB2 is involved

   dli Y/N flag indicating whether any use of DL/I is involved. This includes both DL/I data bases and IMS data communications (IMS DC) calls.

   cics Y/N flag indicating whether CICS is involved

   db2id DB2 subsystem ID to be used during the BIND

   db2col DB2 collection name to be used in creating a package from the DBRM

   wkareanam Name of the work area or driver for which the build is occurring

   otherparms Other parameters could be added in the future.

2. The initialization statements set the names of the MVS host libraries and other variables used in the REXX command file. You might need to change the names of the libraries to match the data set naming conventions for your organization.

3. Determine the names of the temporary data sets that were allocated to various DD statements.
4. Insure that the db2, dli, and cics parameters have only Y or N values.
5. You might need to include logic here to set the values of parameters that control the linkage editor and DB2 bind. These values can be set based on a combination of the work area name and the other parameters (db2, dli, cics and so on) that are received by mvsrplnk.rxp.
6. Call the correct (internal) subroutines based on the parameters that were passed to mvsrplnk.rxp. If DB2 is being used, call getdbrmnams to determine which parts have DBRMs. See “Notes on mvslnk.jcl (the Build Script)” on page 70. for an explanation of the algorithm used. For each part that has a DBRM, call the bind subroutine. After all DBRMs have been bound, close the MSGLIST file that contains all the results of the DB2 commands.
7. Build the concatenation sequence that is used for the linkage editor SYSLIB statement. The concatenation sequence varies depending on whether CICS, DL/I, and/or DB2 are used by the program(s) being linked.
8. Allocate data sets that are used by the linkage editor. The data sets allocated here correspond to the data sets that are used in your current JCL for the linkage editor.
9. Invoke the linkage editor, specifying the parameters to be used. The parameters are hardcoded in this example.
10. If there was an error, call the errorhndlr routine to format the linkage editor messages to go back to TeamConnection.
11. If there were no errors, call the TCOUThndlr routine to save the messages and other information that were written to the SYSPRINT file by the linkage editor.
12. Allocate data sets that are used by the DB2 BIND. The data sets allocated here correspond to the data sets that are used in your current JCL for a DB2 BIND.
13. Remove the values from any previous use of bindctl, bindbctl, and bindactl. These 3 arrays are used to build up the following statements for a CLIST to run the BIND:

    bindctl DSN and BIND statements

    bindbctl Statements in the CLIST that must run before the DSN and BIND statements

    bindactl Statements in the CLIST that must run after the DSN and BIND statements.

    A CLIST is used for the BIND because both the DSN and BIND statements have return codes and there needs to be a way of capturing and checking the return codes from each of the commands.
14. Build the arrays containing the statements needed for the CLIST.
15. Build the CLIST using the 3 arrays.

16. Set up to trap all the output from the CLIST in bindout. Bindout will contain the messages from both the DSN and BIND statements.
17. Run the CLIST.
18. If this is the first BIND, open the MSGLISTX data set that corresponds to MSGLIST in the mvslnk.jcl. Add the output from the DB2 commands to MSGLIST. MSGLISTX is closed by the main routine near note (6) after all DBRMs have been bound.
19. If there was an error, call the errorhndlr routine to format the DB2 messages to go back to TeamConnection.
20. If there were no errors, free the DB2 message output stored in SYSPRINT.
21. Use the INLIST and OUTLIST DD statements as a way of determining which parts are DBRMs. See “Notes on mvslnk.jcl (the Build Script)” on page 70 for an explanation of the algorithm used.

VARIATIONS IN LINK AND BIND

The following sections describe variations in the basic technique that is described above.

“Combining mvsrxpli.rxp with mvsrplnk.rxp”

“Using a BIND Command File” on page 86

“Reducing the Number of Build Messages” on page 89

Combining mvsrxpli.rxp with mvsrplnk.rxp

If each object module is always linked by itself and never linked into other modules, it should be possible to combine mvspli.jcl with mvslnk.jcl and mvsrxpli.rxp with mvsrplnk.rxp. This technique has not been tested because mvsrplnk.rxp supports the more general case where it is necessary to:

Link several object modules together and bind their DBRMs at the same time

Link and bind on a different MVS system from where the precompile, translate, and compile are done.

Provide ability to ship object code and DBRMs as part of a product.

If you decide to combine mvspli.jcl with mvslnk.jcl and mvsrxpli.rxp with mvsrplnk.rxp, consider the following:

Build Tree:

- The .obj and .dbr parts might not need to be included in the build tree if there is no reason to keep them.

- The .msg output file could be eliminated from the build tree -- it was added primarily as a means of matching the .lnk part to remove it from the list of unique part names when determining the DBRM names.

Merging mvslnk.jcl into mvspli.jcl

- The DD statements for LINKCTL and SYSLMOD need to be added to mvspli.jcl
- The DD statements for DBRM and OBJECT should be removed if the .obj and .dbr parts are removed from the build tree. It might be desirable to keep the 2 DD statements and use permanent data sets on the MVS host to save a copy of the object code and DBRM.
- The DD statement for MSGLIST should be removed if the .msg part is removed from the build tree.
- If the .obj, .dbr, and .msg parts are removed from the build tree, the OUTLIST DD statement should use &TCINPUT rather than &TCOUTPUT. This is because the .lkd part has a file type of **None** and thus does not contribute to &TCOUTPUT. Using &TCINPUT means that the .pli and .lnk parts are the only inputs. Use a naming convention that the .pli and .lnk part names must match except for the file extension.

Merging mvsrplnk.rxp into mvsrxpli.rxp

- The parameters for dli, db2id, and db2col should be added to mvsrxpli.rxp.
- The variables plilib, plilib2, plilib3, plilib4, reslib, and linklib that are set in mvsrplnk.rxp should also be set in mvsrxpli.rxp.
- The data set names for LINKLIB and SYSLMOD would need to be determined using the getdsn_last (internal) subroutine.
- The dli parameter should be forced to a Y/N value.
- There is no need to have a loop to handle multiple DBRMs because each PL/I module produces just one DBRM.
- Logic would be needed to reallocate the object module from the compile step to the link edit step and the DBRM from the precompile step to the BIND step.

Using a BIND Command File

Using &TCINPUT and &TCOUTPUT to determine which part names represent a .obj / .dbr pair might not be applicable to your organization. For example, if you have other parts (for example, a PSB, CICS BMS source for a map, or IMS MFS source for a map), adding more parts like .msg to balance out these extra parts can be awkward.

An alternative technique is to use a BIND command file that contains all the DB2 commands that are required to do all the binds for the DBRMs that are involved. This is similar in concept to using the .lnk file for the link edit control statements.

To use a BIND command file, make the following changes:

Create the BIND command files in TeamConnection. For example, for the PL/I program with DB2 (plidb2.pli), create a corresponding BIND command file called plidb2.bnd that contains the following:

```
BIND PACKAGE(TOOLCICS) -
      MEMBER(PLIDB2)  -
      ACT(REP)  -
      VALIDATE(BIND)  -
      ISOLATION(CS)
      ENABLE(xxx)              /  Optional, if use fix           /
      OWNER(xxxxxxx)           /  comment & continuation mark /
```

where TOOLCICS is the DB2 collection ID

In the build trees shown in Figure 16 on page 27 and Figure 24 on page 32, add a bind control file (for example, plidb2.bnd or plidcic.bnd) as input to the .lkd part.

In mvslnk.jcl (Figure 28 on page 70), add the following:

- Add a BINDCTL DD statement for a sequential file that looks like:

```
//BINDCTL    DD  TCEXT=BND,DISP=(NEW,DELETE),
//              UNIT=SYSDA,SPACE=(32   ,(3 ,1 )),
//              DCB=(RECFM=FB,LRECL=8 ,BLKSIZE=32  )
```

- Remove the INLIST and OUTLIST DD statements.

Modify the properties of the 4 builders that do link edits (mvsplilnk, mvsdb2plilnk, mvscicsplilnk, and mvsdb2cicsplilnk) to remove the DB2 collector id from the parameter list. For example, for mvsdb2plilnk, change the parameters on the builder definition:

From:

```
'MVSRPLNK Y N N DSNA TOOLCICS &TCWKAREA'
```

To:

```
'MVSRPLNK Y N N DSNA &TCWKAREA'
```

In mvsrplnk.rxp (Figure 29 on page 74), do the following:

- Change the parse statement at comment (1) to remove the DB2 collection ID. The new parse statement looks like:

```
parse arg db2 dli cics db2id wkareanam otherparms
```

- In the section that retrieves the data sets names, near the end of comment (3), add the statements to retreive the data set name for the BINDCTL DD statement as follows:

```
dsinfo = getdsn_last('BINDCTL')
if dsinfo >   then
   call errorhndlr GETBINDCTL,dsinfo
bindmbr = SYSDSNAME
```

– Change the section that calls the appropriate routines at comment (6) to:

```
call  link
 if db2 = 'Y' then
   do
      call  bind
   end
```

– Change the section that creates the bind control statements at comment (13) to read the statements from the BINDCTL DD statement rather than hard coding them. Change:

From:

```
drop bindctl bindbctl bindactl

/  The following assumes that packages are used and that the bind        /
/  control statements can be in the REXX command file and controlled  /
/  with parameters to the builder                                                    /
bindctl.1    = 'DSN SYSTEM('db2id')'
bindctl.2    = 'BIND PACKAGE('db2col') -'
bindctl.3    =  '        MEMBER('dbrmmbr.i')  -'
bindctl.4    =  '        ACT(REP)  -'
bindctl.5    =  '        VALIDATE(BIND)  -'
bindctl.6    =  '        ISOLATION(CS)'
bindctl.7    = '         ENABLE(xxx)'             /  Optional, if use fix           /
bindctl.8    = '         OWNER(xxxxxxx)'         /  comment & continuation mark /
```

To:

```
drop bindctl binddsn bindbctl bindactl

/  The following assumes that the BIND control statements                  /
/  are in a part maintained within TeamConnection                             /
"ALLOC FI(BINDMBR)    DA('"bindmbr"') SHR REUSE"                                    /
"EXECIO   DISKR BINDMBR ( STEM bindctl. OPEN FINIS )"                          /

/  Create the DSN SYSTEM statement                                                  /
binddsn.1    = 'DSN SYSTEM('db2id')'
```

– Change the statements that build the CLIST to run the BIND commands at comment (15) to also include the binddsn. array. The new statements should look like:

```
/  Build a CLIST that can run the BIND commands                                  /
"EXECIO   DISKW BINDIN ( STEM bindbctl. )"
"EXECIO   DISKW BINDIN ( STEM binddsn. )"         /  <=== new statement        /
"EXECIO   DISKW BINDIN ( STEM bindctl. )"
"EXECIO   DISKW BINDIN ( STEM bindactl. OPEN FINIS )"
```

– Modify the statements that copy the BIND messages to the MSGLIST DD statement at comment (18). The new statements should look like:

```
dsinfo = getdsn_last('MSGLIST')
if dsinfo >   then                    /  Save BIND messages in MSGLIST DD /
   call errorhndlr GETMSGLIST,dsinfo
msglist = SYSDSNAME
"ALLOC FI(MSGLISTX)      DA('"msglist"') SHR REUSE"
"execio   diskw MSGLISTX (stem bindout. open finis)"
```

The if logic to check on whether it is the first bind or not is no longer required because all the binds are done with a single call to DB2.

- Modify the FREE statement just before comment (19) to also free the BINDMBR file.

  ```
  "FREE FI(BINDIN DBRMLIB BINDMBR)"
  ```

- Remove the getdbrmnams routine that begins at comment (21).

If you use crtplibt.cmd as shown in Appendix B, “SAMPLE REXX TO CREATE BUILD TREES” on page 91, modify it so that it connects the .bnd part as input to the .lkd part.

Reducing the Number of Build Messages

If you only want linkage editor messages to be returned when there is an error, replace the calls to TCOUThndlr with statements that free the output listing. The following change is needed to the link step at comment (11):

From:

```
call TCOUThndlr LINK,link_rc,SYSPRINT
```

To:

```
"FREE  FI(SYSPRINT)"
```

APPENDIX B. SAMPLE REXX TO CREATE BUILD TREES

The build trees shown in Figure 12 on page 25, Figure 16 on page 27, Figure 20 on page 30, and Figure 24 on page 32, follow 2 basic patterns - one without DB2 and one with DB2. Therefore, a REXX command file can be used to create the connections for the build tree. The syntax for the command is:

```
crtplibt workareaName pgmname db2

   where workareaName is the name of the work area in which you
                            want to create the build tree
         pgmname          is the name of the PL/I program for which
                            you want to create the build tree
         db2              is a Y/N indicator that indicates whether
                            DB2 is involved

   examples: crtplibt initwa plipgm n
             crtplibt initwa plidb2 y
```

Figure 30 on page 92 shows a sample REXX command file to create the build trees shown in this document.

© Copyright IBM Corp. 1997

```
/                                                    /
/  Create a build tree for a PL/I source part in TeamConnection                  /
/                                                    /
/  TRACE All  /

/  This block of code gets the name of the command, for later use.               /

parse source . . cmdName
cmdName = filespec("name",cmdName)
cmdName = substr(cmdName,1,lastPos('.',cmdName)-1)

/  Parse the command line and display usage, if needed.                              /

parse arg workareaName partname db2

if workareaName = "" | workareaName = "?" then
  do
     say 'usage: 'cmdName' workareaName partname db2'
     say "This procedure creates a build tree for the specified part"
     say "in the specified workarea.    If the db2 parameter is set to Y,"
     say "then the parts needed to support DB2 (the .dbr and .msg) parts"
     say "are also connected to the build tree."
     say "Use just the part name without the extension (for example, plipgm)."
     say "All necessary parts are assumed to exist."
     say "No error checking is included."
     say "Family and release are defaulted from the environment variables."
    exit  1
  end
```

Figure 30 (Part 1 of 2). REXX Command File to Create Build Trees

```
/  Set family and release from the environment variables                    /

family = value('TC_FAMILY',,'OS2ENVIRONMENT')
if family = '' then
  do
     say "The TeamConnection family was not set by the"
     say "TC_FAMILY environment variable."
    exit  1
  end

release = value('TC_RELEASE',,'OS2ENVIRONMENT')
if release = '' then
  do
     say "The TeamConnection release name was not set by the"
     say "TC_RELEASE environment variable."
    exit  1
  end

/  Make the connections required to create the build tree                   /

   'teamc Part -connect 'partname'.pli -parent 'partname'.obj -input' ,
        '-release 'release ' -workarea 'workareaName ' -type TCPart'
   'teamc Part -connect mvsrxpli.rxp -parent 'partname'.obj -input' ,
        '-release 'release ' -workarea 'workareaName ' -type TCPart'
   'teamc Part -connect 'partname'.obj -parent 'partname'.lkd -input' ,
        '-release 'release ' -workarea 'workareaName ' -type TCPart'
   'teamc Part -connect 'partname'.lnk -parent 'partname'.lkd -input' ,
        '-release 'release ' -workarea 'workareaName ' -type TCPart'

   if db2 = 'y' then                        /  if db2 used, more connections  /
     do
        'teamc Part -connect 'partname'.dbr -parent 'partname'.obj -output' ,
            '-release 'release ' -workarea 'workareaName ' -type TCPart'
        'teamc Part -connect 'partname'.dbr -parent 'partname'.lkd -input' ,
            '-release 'release ' -workarea 'workareaName ' -type TCPart'
        'teamc Part -connect 'partname'.msg -parent 'partname'.lkd -output' ,
            '-release 'release ' -workarea 'workareaName ' -type TCPart'
     end

exit
```

Figure 30 (Part 2 of 2). REXX Command File to Create Build Trees

VARIATIONS IN CREATING THE BUILD TREES

You could modify crtplibt.cmd to include the creation of the .obj, .dbr, and .msg parts. Creating these output parts would insure that they are created before the connections are made to create the build tree.

APPENDIX C. GLOSSARY

Access Authority Group. An access authority group is a set of TeamConnection actions.

Access List. An access list is a list of TeamConnection users and their related access authority groups. The access list describes the actions a user can perform upon data controlled by the component or any descendant components. A user has explicit authority to perform these actions. Access can be inherited by other components.

Build Agent. A build agent is software on an OS/2 workstation that monitors the TeamConnection family's job queue for build events. A build agent removes build events from the job queue, sends the build script and input files to a build processor, and stores the returned outputs in the TeamConnection ObjectStore database.

Build Processor. A build processor is software on a machine in a client/server network that waits for a build script and input files from a build agent. A build server runs the build script by passing control as needed to the tools specified in the build script. The build server then sends the outputs back to the build agent.

Build Script. A build script specifies the command(s) to run to build a part. It is similar to one or more steps in an MVS job and can invoke translators such as a DB2 preprocessor, CICS translator, PL/I compiler, or a linkage editor.

Build Server. Build agents and build processors always exist in pairs. A build agent/build processor pair is called a build server.

Build Tree. The structure that represents the configuration of the parts of an application.

Builder. A builder is a TeamConnection object that identifies a build script and the parameters to use when building a part.

Component. A component organizes development data, controls access to the data, and notifies users when certain actions occur. Components can be used to organize the data by application, development team, type of data, or a combination.

Family. All data about an application is stored in a family. Families do not know about each other and cannot share data.

Family Server. The machine that runs the data base and stores/retrieves parts. It is a good idea to have a TeamConnection client on the family server.

© Copyright IBM Corp. 1997

Host List. A list of client machine addresses for access by a user ID. The host list provides the list of physical machines from which a particular user can log on.

JCL. Job Control Language

Parser. A tool associated with a part that runs at build time and examines the part for dependencies (include statements, PL/I include members)

Parts. All data is stored in an object data base. Examples: source programs or include members.

Release. A release is a window into the data base. It controls the application development process. Parts can be linked or not linked between releases. Parts can have the same name in different releases.

Sibling Outputs. Two or more parts that are built together. The build tree diagram for the first shows the second part as an output. The build tree diagram for the second shows the first part as an output. The same builder and parameters apply to both parts.

TeamConnection Client. A machine that runs the TeamConnection client software. The user interface to TeamConnection is through a TeamConnection client.

Work Area. A private view of the parts in a release. Developers make changes to parts that are checked out to work areas.