Systems/C Utilities
Version 2.30

Systems/C Utilities
Version 2.30

Dignus, Systems/C, Systems/C++ and Systems/ASM are registered trademarks of Dignus, LLC.

Contents

How to use this book

This book describes the utility programs provided with the Systems/C and Systems/C++ products and how to use these to create OS/390 programs.

For more information about Systems/C or Systems/C++, see the Systems/C or Systems/C++ manual.

For more information, contact Dignus, LLC at (919) 676-0847, or visit http://www.dignus.com.

Systems/C Utilities Overview

The Systems/C utilities are programs which help create and manage program creation and object libraries.

The utilities include:

PLINK   The Systems/C pre-linker and linker. The pre-linker performs AUTOCALL name resolution to produce the object file that will be linked to become the final executable program, either in a PDS, PDSE or in the Hierarchical File System (HFS). On cross-platform hosts, **PLINK** can also perform the final linking step locally, creating a TSO TRANSMIT file which contains the final OS/390 load module. This avoids any linking on the host.

DPDSLIB   A utility which allows "autocalling" of long symbol names within a PDS on OS/390 or z/OS.

DAR   The Systems/C archive librarian. DAR gathers together object files into a single entity which can be used for pre-linking.

DRANLIB   The Systems/C ranlib utility. DRANLIB creates a symbol definition table in a DAR archive so that AUTOCALL resolution can find the defining instance of a symbol without depending on the DAR member name.

GOFF2XSD   GOFF2XSD converts GOFF objects (generated by the IBM HLASM assembler) into XSD objects.

DCCPC   A CICS Command Processor for C source. Converts "`EXEC CICS`" commands into the appropriate C code to interface with the CICS run-time.

DB2PPC   A DB2 Command Processor for C source. Converts "`EXEC SQL`" commands into the appropriate C code to interface with the DB2 run-time.

D2S   A DSECT to `struct` converter. Converts DSECTs found in assembly ADATA files into C `struct` definitions.

PLINK

Using PLINK

PLINK gathers the input objects together, performing AUTOCALL resolution where appropriate, producing a single file which can then be processed by the IBM BINDER or IBM linker IEWL on older systems. Optionally **PLINK** can also perform the final linking step, producing a TSO TRANSMIT file containing the resulting load module.

Input to **PLINK** can be either object data sets, **DAR** archive libraries, LOAD modules within TSO TRANSMIT files, or control statements. **PLINK** supports GOFF, XOBJ and OBJ data set file formats. GOFF format object data sets must be fixed block with an `LRECL=80`.

When **PLINK** discovers an IBM LE object deck (an appropriately decorated XOBJ file), it performs the functions of the IBM LE pre-linker. Otherwise, as **PLINK** gathers objects, it performs the Systems/C pre-linking functions.

After performing pre-linking functions, if the –*b* option is specified (indicating "binding"), **PLINK** performs the functions of the IBM BINDER, resolving any remaining unresolved symbols, performing relocations and producing the resulting TSO TRANSMIT module which describes the load module. If the –*bonly* option is specified, no pre-linking functions are applied.

When generating a TSO TRANSMIT file, **PLINK** will replace just the named member (specified by –*syslmod*), preserving any other pre-existing LOAD module members.

For Systems/C pre-linking, **PLINK** examines the defined symbols, looking for a Systems/C initialization script section. Furthermore, if the –*fnosname* option was specified on a **DCC** or **DCXX** command line, **PLINK** provides unique section names for the object. During this process, **PLINK** will produce warnings on discovering any duplicate definitions or duplicate section names.

At the end of the generated file, **PLINK** will append a section containing pointers to the located initialization script sections, which the Systems/C run-time library will use for re-entrant initialization. Also, if needed for C++, **PLINK** will generate

the appropriate C++ constructor/destructor tables and place that in the resulting object.

If **PLINK** discovers any Dignus debugging information embedded in the object deck, **PLINK** will gather this information together and apply appropriate transformations to make it ready to use by a debugger. **PLINK** will then generate a debugging information file that is associated with the eventual program (controlled by the *–dbg=filename* option).

On cross-platform hosts, **PLINK** may be employed to gather the objects together into one file for transferring to the mainframe for the final link step, or can perform the final linking itself, producing a load-module embedded in a TSO TRANSMIT file.

Unless the *–b* or *–bonly* option was specified, the output of **PLINK** is then intended to be processed by the IBM BINDER or linker to produce the executable load module. By default, the output format will be the same as the input format (either OBJ or GOFF), except that XOBJ is converted to GOFF (controlled by the *–goff / –esd / –xsd* options).

PLINK options

PLINK supports the following options. Any remaining command-line parameters are treated as input file names. Input files must be OBJ object decks, XOBJ object decks, GOFF object decks, **DAR** archives, or TSO TRANSMIT files of LOAD libraries.

Also, on Windows, z/OS and OS/390 host platforms, **PLINK** supports the *–@filename* option. The contents of the specified file are inserted in the command line at the point where the *–@filename* appears. This provides a mechanism for specifying command lines which are longer than the operating system supports.

–ac=*n* Specify the authorized program facility (APF) authorization code of the final executable, specified in the TSO transmit file, when the *–b* or *–bonly* option is enabled. The authorization code *n* must be an integer between 0 and 255. The default value is 0.

–allow_ref=*name* The *–allow_ref* option defines a name that **PLINK** will not consider unresolved after all processing. After all AUTOCALL processing is performed, **PLINK** produces a list of unresolved references, and appropriately sets the return code. Symbol names specified via the *–allow_ref* option will not appear in this output, or affect the return code. Note, however, that *–allow_ref* does not provide a definition of the symbol, thus **PLINK** will continue to try and resolve any references to the *name* via normal AUTOCALL processing.

–amode=*amode-setting* Specify the `AMODE` setting of the final executable when the *–b* or *–bonly* option is enabled. *amode-setting* can be either `any`, `min`, `24`, `31` or `64`.

–asm   To be used with *–shared*, this specifies that it is step #1 of building a shared library, to produce the `sodef.asm` and `soref.asm` files.

–b   After performing normal pre-linking functions, **PLINK** will "bind" the objects, producing a load module. On cross-platform hosts, this causes **PLINK** to produce (or update, if it already exists) a TSO TRANSMIT file which can then be reinstated with the TSO RECEIVE command on the mainframe platform.

–blksize=*N*   Sets the block size of the resulting load module when the *–b* or *–bonly* options are specified. If *–blksize* is not specified, the default block size is 27998, the optimum block size for a 3390 device.

–bonly   Similar to *–b* except that normal pre-linking functions are not performed. Only the "binding" step is performed and a TSO TRANS-MIT file is produced. This can be helpful if the object has been previously processed by **PLINK**.

–bvse   Perform the normal steps for the *–b* option, but generate a VSE "unloaded PHASE" instead of a TSO TRANSMIT file.

An "unloaded PHASE" is an object deck decorated with a starting description card. Almost all relocation processing has been performed so that the object deck can simply be loaded to generate the PHASE on VSE.

The VSE linker does not support relocations larger than 4 bytes. In this case, **PLINK** will truncate the relocation and adjust the offset so that it becomes a 4-byte relocation. If the relocation includes a negative action, **PLINK** will generate a warning, as subtractions may not underflow correctly in this situation.

–cmap*[=file]*   Causes **PLINK** to produce a report of the CSECT names assigned to input objects which were compiled with the *–fnosname* option. When *–fnosname* is used, **PLINK** assigns a unique name to the various sections produced by the compiler. This map can be helpful when referring back to particular objects based on the run-time section name. If *=file* is specified, the map is written to the specified file name, otherwise it is written to the `stdout` file stream.

–dbg=*filename*   Specifies the name of any **PLINK** produced debugging information file. If no *–dbg=* option is specified, **PLINK** uses a default name.

The default name for the debugging information file on OS/390 and z/OS is "`//DDN:SYSDBG`", on cross-platform hosts it is "`p.dbg`".

–DD=*NAME=template* Specify a "DD" definition to use for search for `INCLUDE` cards. The *NAME* specified is a DD-name that might appear as

a PDS name on an INCLUDE card. The *template* specified is a search template which is expanded with the member name. The search template rules are the same as those used for AUTOCALL name searches. The *template* may be the name of an archive or TSO TRANSMIT file. See the section on AUTOCALL searches for more information about template expansion.

The section on `INCLUDE` statement processing describes the use of the *–DD* option further.

–esd   Generate output in object module format with symbols defined by `ESD` cards. Symbol names longer than 8 characters will be silently truncated (unless they are remapped by another option like *–p*).

–except=*name1,name2,...* Specifies names that are exceptions to the "rename all" rule when the –renameall option is specified.

The –renameall option renames all symbols except for names specified in the –except options.

More than one name can be specified, each name is separated by commas.

The –except option can be specified multiple times.

–exclude_allrefs   Specifies that no external shared references will be generated in the `sodef.asm` file. That way, after linking a shared library, there is no further AUTOCALL attempt to resolve those symbols. That AUTOCALL attempt usually fails and, in addition, makes it hard to later determine which shared library was responsible for an undefined symbol reference.

–exclude_refsfile=*filename* Specifies a file which contains a list of symbols which will not have references generated in `soref.asm`, when **PLINK** is run with *–shared –asm*. The file is generally going to be generated by *–sodefnames=filename*. In this way, an autocall library of shared library stubs can be generated (instead of an SO object) that does not contain any expensive external shared references for symbols which are defined locally to the shared library.

When *–sodefnames* is used to create the file, **PLINK** should be run with all of the objects in the shared library specified on the command line, so that all of the defined symbols are known. Then *–exclude_refsfile* is used when running **PLINK** on individual members of the shared library, so that **PLINK** will know which symbols are defined in the other members of this same library.

–exclude-symbols=*symbols* The symbols which are specified (in a comma-separated list) will not have any external definitions generated for them in the `sodef.asm` file when **PLINK** is run with *–shared –asm*. This allows certain symbols to be treated as global within the library, but not exported to be visible outside of the shared library.

–fdaranycase   When **PLINK** performs autocall resolution using **DAR** archives, if there is no **DRANLIB**-generated symbol table, in the archive, **PLINK** uses the name of the member as the definition name. In this case, **PLINK** matches the reference to the **DAR** member name, much as PDS-based lookup would be performed in a traditional z/OS environment.

By default, **PLINK** treats the member names and reference names as case-specific, matching only references with the exactly specified **DAR** archive member name.

If the –fdaranycase option is specified, **PLINK** will ignore case differences on this match.

This can better approximate the PDS lookup that is performed on z/OS, where the names are all converted to upper-case by default.

–fdllname=*DLLNAME* When pre-linking IBM DLL objects, the –fdllname option specifies the name of the resulting DLL. It is similar to the `DLLNAME` option available on the IBM pre-linker.

If no –fdllname option is specified, **PLINK** uses the name `TEMPNAME`.

–fkeepweaktxt   The **Systems/C** and **Systems/C++** compilers allow for the multiple definition of `weak` functions. This feature is frequently employed by **Systems/C++** to accomplish automatic template instantiation.

PLINK handles duplicate `weak` function definitions by appropriately adjusting or deleting duplicate `ESD` entries and adjusting `RLD` relocation entries. Further **PLINK** actually removes the function code bytes from the resulting object file, creating a much smaller resulting object file for final linking.

There could be instances, however, where it would be desirable to retain the function code bytes in the resulting object. If *–fkeepweaktxt* is specified, **PLINK** will appropriately adjust `ESD` and `RLD` entries, but will not delete the duplicate function code. The code will remain in the resulting object, but will not be referenced in any fashion.

–fmrc   Specifies that mainframe return code styles are to be used. This is the default on all platforms, but may change in the future. When –fmrc is enabled, **PLINK** returns a 0 for no warnings/errors, 4 if any warnings were generated and 8 or greater for errors.

–fmapdata=*value* Specify a value which is passed as a character string to the `map` function.

–fmapfunc=*location* On Windows and UNIX-style platforms, the *–mapfunc* option provides the name of the DLL or shared-library to load that implements the `map` function.

The DLL (or shared library) should have an entry point named `map` that **PLINK** will invoke to determine name mapping in the resulting object.

On z/OS, this specifies the name of the load module which is `LOAD`'d. The entry-point for this load module implements the `map` function, using standard z/OS 31-bit OS linkage.

PLINK will invoke the `map` function for each symbol name that needs to be mapped from a long name to a short name. If the *–frenameall* option is specified, this function will be invoked for all symbols. The `map` function provides the new symbol name (in EBCDIC) for **PLINK** to place in the resulting object.

The `map` function is defined as:

```
void
map(int flag, void **space, char *user_parm,
    const char *orig_name, const int orig_name_len,
    char **result_name,
    int *result_name_len);
```

flag

The *flag* parameter is an integer value that passes indicators to the `map` function.

The lower two bits of the *flag* value have the following meanings:

`0x01`	Initial "set up" invocation.
`0x02`	Final "termination" invocation.
`0x03`	File name mapping request.

When the lower two bits of *flag* have the value `0x01` or `0x02`, indicating "set up" and "termination", the other parameters are ignored.

When the lower two bits of *flag* have the value `0x03` the source name to map is based on the input file name. **PLINK** uses the mapped name value to create PLINK-generated names for various purposes, mostly related to C++ facilities.

When the lower two bits of *flag* are `0x00`, *flag* will then indicate the type of symbol being mapped. In this situation, *flag* will have the values:

`0x04`	Symbol is an SD (Section Definition)
`0x08`	Symbol is an ED (Element Definition)
`0x0C`	Symbol is an LD (Label Definition)
`0x10`	Symbol is an PR (Psuedo Register Reference)
`0x14`	Symbol is an ER (External Reference)

space

space is a pointer to a (`void *`) pointer which can be useful for the `map()` function to pass data to

subsequent invocations. This pointer will be maintained across invocations of the `map()` function. Thus, the function can allocate space that will be available to subsequent invocations.

user_parm	*user_parm* is a character pointer that provides the value specified in the *–fmapdata* **PLINK** option.
orig_name	*orig_name* is a pointer the original symbol name.
orig_name_len	*orig_name_len* is the number of characters in the original symbol name.
result_name	This is a pointer to a character pointer, thus it should be set to the address of a buffer containing the name **PLINK** should place in the object deck.
result_name_len	This is a pointer to an integer which contains the number of characters in the *result_name.*

–fmsgtostdout   Normally, **PLINK** writes error and warning messages to the `stderr` file stream. *–fmsgtostdout* causes the messages to be written to the `stdout` stream. This can be helpful on some Windows platforms that don't allow redirection of the `stderr` stream.

–fnomrc   Specifies that UNIX (non-mainframe) return code styles are to be used. In a future release, this may become the default for cross-platform hosts (currently *–fmrc* is the default on all platforms.) When *–fnomrc* is enabled, PLINK returns a 0 even if any warnings were generated. If an error is generated, **PLINK** will return a non-zero value. This is helpful when incorporating **PLINK** into cross-platform build facilities (e.g. `make`) that expect a zero/non-zero return code.

–fquiet   The *–fquiet* option causes plink to suppress all diagnostic output. Only error messages will be generated.

–from=*node.uid*   Specify values for the from node name (`INMFNODE`) and from UID (`INMFUID`) headers in TSO TRANSMIT files. The UID is optional. The default is `DIGNUS.PLINK`. This option only has an effect when combined with *–b* or *–bonly.*

–fsplitdata=*value*   Specify a value which is passed as a character string to the `splitnm` function.

–fsplitfunc=*location*   On Windows and UNIX-style platforms, the *–splitfunc* option provides the name of the DLL or shared-library to load that implements the `splitnm` function.

The DLL (or shared library) should have an entry point named `splitnm` that **PLINK** will invoke to determine the output "split" object name.

On z/OS, this specifies the name of the load module which is `LOAD`'d. The entry-point for this load module implements the `splitnm` function, using standard z/OS 31-bit OS linkage.

PLINK will invoke the `splitnm` function for each new object deck it needs to create. `splitnm` computes the resulting name of that object deck, or if that object deck should not be written, it returns zero.

The `splitnm` function is defined as:

```
int
splitnm(int flag, void **space, char *user_parm,
        int id, char *orig_name, char *member_name,
        unsigned char *result);
```

flag
: The *flag* parameter is an integer value that describes the kind of input file. *flag* can have any of these values logically OR'd together:

0x04	The file is a primary input file
0x08	The file is a primary include file
0x10	The file is a secondary include file
0x20	The file was included via autocall name resolution
0x40	The file was created by **PLINK**
0x40000000	The file is a LOAD module (member of a TRANSMIT module)
0x80000000	The file was a member of a **DAR** archive

: *flag* will be `0x01` for the initial "set up" invocation, and `0x02` for the final "termination" invocation.

space
: *space* is a pointer to a (`void *`) pointer which can be useful for the `splitnm()` function to pass data to subsequent invocations.

user_parm
: *user_parm* is the character pointer that provides the value specified in the –*fsplitdata* **PLINK** option.

id
: *id* is an integer identified assigned by **PLINK** to the input object file.

orig_name
: *orig_name* is the original input file name

member_name
: If the *flag* value's upper bit (0x80000000) is set, then *member_name* specifies the member name of the **DAR** archive. If the 0x40000000 bit is set then *member_name* specifies the member within the TSO TRANSMIT module.

result
: *result* is a pointer to a buffer of 1024 bytes that on return should contain the file name where **PLINK** will write the output file.

The return value from the `splitnm` function indicates if the file should be produced or not. If the return value is zero, then the output is not generated.

The `splitnm` function is invoked with two "special" *flag* values indicating initial start-up and termination. If the value of *flag* is `0x01` then this is the initial invocation of the `splitnm` function. In this case the other arguments are invalid and intended to be ignored. This is provided so the `splitnm` function can perform whatever preparation tasks might be needed for proper function. Similarly, if the *flag* value is `0x02`, this is the terminating invocation, so that `splitnm` can perform any termination functions it may require. As with the initial invocation, when *flag* is `0x02`, the other parameters should be ignored.

–fsplitloc=*location* Specify the location in which to write the component object decks.

On Windows and UNIX-style platforms, this is a directory. On z/OS this is a PDS.

–fsplitobjs=*file* The *–fsplitobjs* option indicates that **PLINK** should "split" the conglomerate output into it's constituent pieces. The combined object deck, including any **PLINK**-generated portions is split into individual object files based on the input object deck names.

This option is not available if the *–b* or *–bonly* option is specified.

–fsysdefsd=*file* Specify the name of the **PLINK**-generated definition file when prelinking IBM DLL objects. This is the same as the `SYSDEFSD DD` specified on the IBM prelinker.

When pre-linking IBM DLL objects that export symbols **PLINK** will create the appropriate cards and write them to the named file.

If *–fsysdefsd* is not specified, **PLINK** writes the DLL definitions to a file name "`sysdefsd`".

–goff Generate output in Generalized Object File Format (GOFF). GOFF output is capable of representing all of the information from any of the input object formats.

–I*template* Provide a path to search for `INCLUDE` cards that do not specify a DD. The *template* will be expanded with the `INCLUDE`d name. An archive or TSO TRANSMIT file may be specified for searching. This option may be used any number of times.

–l*ar* Add an archive file *ar* to the list of files to link. This option may be used any number of times. **PLINK** will search its *–L* path-list for occurrences of the file named `lib`*ar*`.a` for every ar specified.

–L*searchdir* This command adds the path *searchdir* to the list of paths that **PLINK** will search for **DAR** archive libraries. This option may be used any number of times.

–map	Causes **PLINK** to produce a load-module offset map when the *–b* or *–bonly* option is enabled.
–noprem	Disables **PLINK**'s processing of DXDs, CXDs and Q-cons. See the *–prem* option for more details.
–noxmit	Normally, when the *–b* or *–bonly* options are specified, **PLINK** creates a TSO transmit file containing the resulting load module. The *–noxmit* option causes **PLINK** to simply write the "raw" load module records. When *–noxmit* is specified, the resulting file is not in TSO transmit format.
	Note that the load module records do not contain the module's starting offset and other information normally stored in the load module's PDS directory entry.
–o*file*	The *–o* option specifies the name of the output *file*. On cross-platform hosts, **PLINK** writes to the file `p.out` by default; on OS/390 and z/OS **PLINK** writes to the file `//DDN:SYSMOD`.
	If the *–b* or *–bonly* options were specified, the output file is a fully realized load module in TSO TRANSMIT format.
–p	The *–p* option causes **PLINK** to process the incoming files, converting any long symbol entries to 8 characters. When a symbol entry longer than 8 characters is discovered, **PLINK** replaces it with a name of the form ST*nnnnnn*, where *nnnnnnnn* is a unique identifier given to the name. If the ultimate target of the load module is a PDS, and not a PDSE, then using the *–p* option will properly shorten the incoming names so the IBM binder will be able to properly link into the PDS. The **PLINK** output will display long names that are mapped to shorter ones.
	An *x* may be appended to the *–p* option. As well as converting long symbols to unique shorted names, specifying *–px* causes **PLINK** to convert XSD or GOFF object cards into ESD cards. Older versions of the IBM linker only handle ESD-type input. Specifying *–px* cause **PLINK** to generate output suitable for these older linkers.
	As well, a *u* may appened to the *–p* option which alters the mapping applied. By default **PLINK** only maps names which are longer than 8 characters. When *u* is specified in the *–p* option, **PLINK** will map any names that are longer than 8 characters, or which have lower-case EBCDIC letters in them. Furthermore, **PLINK** will attempt to construct a short-name based on the original name. It will use the first characters from the original name (up to 8) and upper-case those. Also, any underscore character (_) will be converted to an at-sign (@). If this contructed name does not already exist, then that will be the chosen "short name". If the construct name does already exist (because it originally existed, or because a previously constructed name is already in place), then PLINK will create a "short name" using the ST*nnnnnn* mapping.

The *u* and *x* can appear in any order.

Note that when ST*nnnnnn* is used the `@ST` portion can be altered with the *–prefix* option.

–prefix=CCC   Typically, when renaming long names to unique short names, **PLINK** composes a name with a 3-character prefix (`@ST`) followed by 5 digits.

The *–prefix* option can be used to specify an alternative prefix. Up to 3 letters can be specified. The first letter should be a character, or an '@' character, or a '#' character.

This can be helpful if the resulting objects are intended to be linked with another runtime environment. For example, the IBM runtime environment, since the IBM pre-linker also uses the prefix `@ST`.

–prem   The *–prem* option causes **PLINK** to internally process `DXD` definitions and handle any `CXD` and `Q`-con relocations. The gathered `DXD` definitions are collectively called the *Pseudo Register Vector* or *PRV*. **PLINK** will gather `DXD` definitions into the `PRV`, assigning offsets to each. Then, any `Q`-cons which reference the elements of the `PRV` will be replaced with their offset. Any `CXD` relocations will be replaced with the total size of the `PRV`. **PLINK** will also produce a table indicating each `DXD` discovered, and its assigned offset. The `PRV` is used by the Dignus compilers to produce re-entrant code.

In order to properly produce the `PRV` table, all of the objects that contain `DXD`s, `CXD`s and `Q`-cons should be presented as input to **PLINK**.

The *–noprem* option can be used to disable this processing, if *–noprem* is specified, **PLINK** passes the `DXD` definitions and `Q`-con and `CXD` relocations through to the resulting object file.

Older IBM linkers, particularly for CMS and VSE, do not properly handle a `PRV` larger than 4K bytes. Thus, when generating code for these platform, *–prem* should be used.

The *–prem* setting is the default.

–r   Allow for re-running **PLINK** on the resulting object. If *–r* is specified, the result from **PLINK** will have all possible AUTOCALL operations performed, with no other processing. The resulting file can subsequently be reprocessed by **PLINK** to perform the other, normal processing. When *–r* is specified, the *–p* and *–px* options will be ignored. Also, any unresolved references will not be diagnosed, as it is assumed they will be resolved in a subsequent **PLINK** step.

–refresh   Specify that the `REFRESHABLE` flag of the final executable should be set when the *–b* or *–bonly* option is enabled. *–refresh* may be abbreviated to *–refr*.

–renameall   The *–renameall* option causes **PLINK** to rename all symbols in the generated object. This can be helpful when linking Systems/C DCALL modules with other runtime environments where the names might clash. For example, when linking a Systems/C DCALL'd object with IBM C/C++, or SAS C/C++.

The names chosen consist of a 3-character prefix, followed by a 5-digit number. By default, the 3-character prefix is `@ST`, but can be altered with the *–prefix* option. The *–fmapfunc* option can be used to provide a user-written function that controls the name mapping.

When *–renameall* is specified, all symbols are renamed except for any specified via the *–except* option.

Thus, a typical approach would be to determine which symbols are meant to be visible to the calling program. Specify these names in the *–except* list, and specify the *–renameall* option. The resulting **PLINK**-generated object deck will have all of the symbols renamed except for the ones that need to be visible to any calling environment.

–rent   Specify that the `RENT` flag of the final executable should be set when the *–b* or *–bonly* option is enabled.

–reuse   Specify that the `REUSE` flag of the final executable should be set when the *–b* or *–bonly* option is enabled.

–rmode=*rmode-setting*   Specify the `RMODE` setting of the final executable when the *–b* or *–bonly* option is enabled. *rmode-setting* can be either `any` or `24`.

–S*template*   The *–S* option specifies a template which is expanded with the AUTOCALL reference name. Any `&m` and `&M` characters found in the template are replaced with the reference name, `&m` indicates that the lower-case version of the name should be used; `&M` indicates upper-case. On OS/390, **PLINK** has a default *–S* option of `-S//DDN:SYSLIB(&M)`. Note that *–S* templates are automatically added to the *–DD=SYSLIB* list.

–shared   Specifies that **PLINK** is being used to create a shared library. If *–asm* is also specified then it is producing the `sodef.asm` and `soref.asm` files created in step #1. Otherwise, it is producing the shared library module created in step #2.

–showctl   Indicate that **PLINK** should provide information messages about any control cards encountered in the input. **PLINK** will produce messages of the form

        `plink: info: control card:` *text-of-card*

where *text-of-card* is the text as it appeared on the input control card.

–sodef=*filename* Provides an alternate filename for the `sodef.asm` file generated with
–shared –asm.

–sodefnames=*filename* Provides a filename into which the list of all defined symbols
will be written if **PLINK** is run with *–shared –asm.* This file is
often created for later use with *–exclude_refsfile=filename* to avoid
generating external references to symbols which are defined within
the current shared library.

–solstub=*name* Provides an alternative to `DCCLSTUB` for the name of the assembly
macro which is used to produce shared library stub routines.

–soname=*name* Provides the name of a shared library, which must correspond to
the load module's name.

–soref=*filename* Provides an alternate filename for the `soref.asm` file generated with
–shared –asm.

–syslmod=*SSS* Specify the resulting output PDS name when the *–b* or *–bonly* op-
tions are enabled. The name *SSS* should completely specify the PDS
name on the mainframe host. A member name can also optionally
be specified.

–t Normally, **PLINK** will issue diagnostic messages regarding dupli-
cate definitions of symbols and sections. The *–t* option suppresses
these messages. In any case, these duplicate definitions are not ad-
dressed by **PLINK** and are simply passed on to the IBM linker.

–to=*node.uid* Specify values for the to node name (`INMTNODE`) and to UID (`INMTUID`)
headers in TSO TRANSMIT files. The UID is optional. The default
is `DIGNUS.PLINK`. This option only has an effect when combined with
–b or *–bonly.*

–u The *–u* option causes **PLINK** to adjust the return code when unre-
solved references are discovered. Normally, after **PLINK** process-
ing; if there remain any unresolved references, the return code will
be 8. The *–u* option causes the return code to be 4 when unresolved
references are encountered. This can be helpful if there are unre-
solved references at the **PLINK** step, which will be later resolved
by the IBM BINDER.

–uu The *–uu* option operates the same as the *–u* option except that the
return code will not altered by any unresolved references. **PLINK**
will continue to generate warnings regarding unresolved references,
but they will not set the return code to 4.

–v Causes **PLINK** to produce the banner indicating the version and
copyright information, then to exit with a return code of 0.

–xref Causes **PLINK** to produce a load-module cross-reference listing
when the *–b* or *–bonly* option is enabled.

–xsd   Generate output in extended object module format (XOBJ) with symbols defined by `XSD` cards. This allows the representation of long names, but does not support all of the features of GOFF. The IBM binder has known bugs with XOBJ input, so PLINK will never output XOBJ decks unless *–xsd* is specified. If an object comes in as XOBJ, PLINK will automatically convert it to GOFF.

–z   The *–z* option is used to change **PLINK**'s processing of **DAR** archives and TSO TRANSMIT files. `-z` is followed by either `allextract` or `defaultextract`. In the default mode of operation, **PLINK** only extracts members which satisfy external references. If `-zallextract` is specified, **PLINK** will extract all members of subsequent archives found on the command line, essentially treating all the **DAR** archive and TSO TRANSMIT members as primary input files. `-zdefaultextract` is used to return to the default extraction method.

PLINK control statements

PLINK recognizes the `ARLIBRARY`, `IMPORT`, `INCLUDE INSERT`, and `RENAME` control statements in the input stream. When the *–b* or *–bonly* options are enabled, **PLINK** will also recognize `ENTRY NAME` and `ORDER` cards used for describing load modules.

If *–b* and *–bonly* are not specified, **PLINK** will copy all other control cards to the resulting output file. All such unrecognized control statements will be gathered together, and placed at the end of the **PLINK**-generated output.

`ARLIBRARY` *name*   Adds name to the list of **DAR** archive libraries **PLINK** will examine to resolve external reference.

`ENTRY` *name*   When *–b* or *–bonly* is enabled, this defines the resulting load module entry point. When the binding options are not enabled, this is copied to the resulting output file.

                An entry point specified on an `ENTRY` card overrides any entry point specified on an `END` card in the intput object.

                If multiply `ENTRY` cards appear in the input, the last one discovered is used for specifying the entry point of the load module.

                If no `ENTRY` card is specified, **PLINK** will use an entry point specified on an `END` card, or if no `END` cards specify an entry point, **PLINK** will use the first byte of the first control section.

`IMPORT` *type dll-name import-name* Used when pre-linking IBM DLL objects. The `IMPORT` card defines a datum/function that can be found in an IBM DLL. *type* specifies the type of reference, either `DATA` or `CODE`. *dll-name* is the name of the DLL module that resolves the reference. *import-name* is the name of the DLL symbol.

`IMPORT` cards are generated by **PLINK** when pre-linking IBM DLL objects. These generated cards are then employed when the symbols in the DLL are to be referenced.

INCLUDE *name*   Causes **PLINK** to include the object file *name* in its processing. On cross-platform hosts, **PLINK** handles `INCLUDE` cards of the form:

> `INCLUDE   NAME( MEMBER)`

in a manner designed to make transition from the mainframe easier.

PLINK will first look for `DD` entries (from any *–DD* options) that specify the name *NAME*. If they are found, **PLINK** will expand the template specified in the *–DD* to search for the *MEMBER*. It will also search for corresponding members in any archive or TSO TRANSMIT files that were provided with *–DD*.

If no *–DD* option specifies the *NAME*, then **PLINK** will consider *NAME* to be the name of an environment variable. This environment variable will be taken as a template value, just as if a *–DD* option had specified it.

If the *NAME* is not specified in the environment and is not specified by a *–DD* option, **PLINK** will attempt to open the file exactly as it appears on the `INCLUDE` card.

If an `INCLUDE` card does not specify a `DD`, such as

> `INCLUDE   MEMBER`

then **PLINK** will search the templates provided by *–I* instead of a `DD`.

Note that on MVS and CMS hosts, **PLINK** simply uses the standard OS-provided `DD`s and does not use the *–DD* options.

For example, on a Windows platform, if **PLINK** discovered the input card:

> `INCLUDE MYPDS(MYOBJ)`

and the option `-DD=MYPDS=C:\DIR1\&M.obj;C:\DIR2\&M.obj;foo.xmi` was specified, **PLINK** would first try to open the file named `C:\DIR1\MYOBJ.obj`. If that file was not available, **PLINK** would then try to open the file named `C:\DIR2\MYOBJ.obj`. If that file was not available, it would look for a matching member within `foo.xmi` (a TSO TRANSMIT file).

This approach allows for a mapping of DD-style concatenations to cross-platform systems; without changing the input to the pre-linker.

For more information about how search templates are expanded, see the section on AUTOCALL processing.

INSERT *symbol*   Causes **PLINK** to add *symbol* as an external reference. If symbol has not been resolved when the primary input has been processed, AUTOCALL processing will attempt to resolve it. This card is also passed to the **PLINK** generated output.

NAME *name*   When –*b* or –*bonly* is enabled, this defines the name of the resulting PDS member. When the binding options are not enabled, this card is copied to the resulting output file.

ORDER *name*   When –*b* or –*bonly* is enabled, this card specifies that the named CSECT should appear first in the output load module.

The effect of the ORDER card is that the given section is moved to the top of the output list. A subsequent ORDER card will place the section named there, before any others specified by ORDER.

The specification (P) can follow the *name* on the order card, indicating the given section is to be aligned on a 4K boundary (page aligned) in the output load module.

RENAME *longname shortname* Used to rename a symbol, providing a specific shortname for a long-named symbol. *longname* is the original symbol name; *shortname* is the resulting shortened name that will be used in the output object module. If the optional SEARCH keyword follows the *shortname* then the short name will be added to the list of unresolved references for AUTOCALL processing.

The RENAME card is only valid when performing IBM pre-linking functions. With normal Systems/C-style objects, the RENAME card will be copied to the output file.

SETCODE AC(*authorizationcode*) The SETCODE statement specifies the authorization code for the generated load module.

The *authorizationcode* specifies an integer value between 0 and 255.

If the –*b* or –*bonly* options is specified (indicating "binding"), then *authorizationcode* will be used as the authorization code for the load module, otherwise the SETCODE command is copied to the output for later processing.

PLINK autocall processing

PLINK will process INCLUDE cards and produce a list of external references. **PLINK** examines each object that participates in a program and it will perform AUTOCALL processing to resolve any external references discovered in the primary input files.

During AUTOCALL processing, **PLINK** examines the list of unresolved external references, looking in the libraries specified with –*S* or –*DD=SYSLIB* for objects that will resolve the references. When such an object is found, it is examined for further

references and for any re-entrant initialization sections. This process continues until all references are resolved, or determined to be unresolvable.

PLINK can employ either a direct file name approach to resolving external references, or take advantage of **DAR** archives, or use LOAD modules within TSO TRANSMIT files, or any combination of these. **DAR** archives, TSO TRANSMIT files, and –*S* specified templates are examined in the order they were discovered by **PLINK**, first on the **PLINK** command line and then in any `ARLIBRARY` control statements discovered in the input.

To search in locations specified by the –*S* option, **PLINK** simply expands the various specified templates with the reference name, and attempts to open the given file. If the open is successful, the file is processed, and the reference is considered resolved. If the file doesn't actually resolve the reference, **PLINK** will produce a warning message.

On hosts which have case-sensitive files systems (UNIX, OS/390 and z/OS), first an exact match of the reference name is tried; then either the upper-case or lower-case version is tried per the `&M` or `&m` characters in the template specification.

PLINK examines **DAR** archives by first looking for any **DRANLIB** generated symbol table member. If the symbol table member is present, **PLINK** uses the symbol table information to determine which member file resolves an external reference. Thus, **PLINK** is able to directly relate any file and its definitions. On hosts that support it, **PLINK** locks **DAR** archives as they are discovered. This prevents accidental changes to the **DAR** archive during **PLINK** processing. Because of this, a **DAR** archive should not be specified more than once on the **PLINK** command line.

When a symbol table member is not present, **PLINK** uses the member in the archive with the same name as the unresolved symbol to attempt to resolve references, in a fashion similar to examining a PDS for a particular member. Note, however, that the file names in **DAR** archives are case-specific and thus **PLINK** uses a case-specific comparison in this case.

When searching a TSO TRANSMIT file, **PLINK** searches for a member with the uppercased 8-character version of the symbol name, as though it was searching a PDS without a table-of-contents file.

On OS/390 and z/OS, **PLINK** has two approaches to searching for PDS members which resolve a reference. If the **DPDSLIB** utility has created a table-of-contents file in the PDS, **PLINK** will use that file to determine which member resolves a reference. The table-of-contents file contains information regarding all of the symbols in each object file in the PDS. Thus, **PLINK** can use the table-of-contents to determine if an object resolves long names which cannot be represented in the PDS directory. If the table-of-contents file is not present in the PDS, **PLINK** will try and open the file name which is the same as the reference. See the section on **DPDSLIB** for more information regarding the table-of-contents member.

Running PLINK

On cross-hosted platforms (Windows and UNIX), **PLINK** is typically executed with the object files listed on the command line; and a –S option or library names to locate any required library objects.

For example, on a Windows platform the command:

```
plink -SC:\sysc\lib\objs_rent\&M prog.obj
```

will read the initial input file, `prog.obj` and examine the `C:\sysc\lib\objs_rent` directory for any AUTOCALL references. Because no –o option was specified, the resulting object file is written to the file `p.out`.

This command, on UNIX platforms:

```
plink t1.obj t2.obj libone.a -L../mylibs -ltwo
```

will read the two primary input objects `t1.obj` and `t2.obj`. It will try and resolve references from the **DAR** archive `libone.a` and then the second **DAR** archive `../mylibs/libtwo.a`.

On OS/390 and z/OS, when run from TSO or "batch" JCL, **PLINK** operates similar to the IBM pre-linker. The resulting gathered object is written to the file `//DDN:SYSMOD` unless otherwise specified. **PLINK** has a default library template of `-S//DDN:SYSLIB(&M)` which causes it to look in the `SYSLIB` PDS for autocall references, other input objects, –S library templates or **DAR** archives may be added in the PARMS option on the **PLINK** step. **PLINK** reads the file `//DDN:SYSIN` as the initial input file. Typically, this file contains `INCLUDE` cards to include the primary objects for the program. Other primary input files may be included in the PARMS for **PLINK**.

For example, the following JCL reads the object `INDD(PROG)` and uses `DIGNUS.LIBCR` as the autocall library:

```
//PLINK EXEC PGM=PLINK
//STEPLIB DD DSN=Systems/C load library,DISP=SHR
//STDERR  DD SYSOUT=A
//STDOUT  DD SYSOUT=A
//SYSLIB  DD DSN=DIGNUS.LIBCR,DISP=SHR
//INDD DD DSN=mypds,DISP=SHR
//SYSIN DD *
 INCLUDE INDD(PROG)
//SYSMOD DD DSN=myoutput.obj,DISP=NEW
```

Note that the `STDERR` and `STDOUT` DDs were specified for **PLINK**'s message output. Also, the `ARLIBRARY` control card could have been used to add additional **DAR** archive files for resolving external references.

When executed under the OpenEdition shell, **PLINK** operates as it does on any other UNIX platform.

Using PLINK to pre-link OpenEdition programs

Systems/C programs can be linked into the Hierarchical File System (HFS), producing a program that can be executed just as any other OpenEdition program.

To create an HFS load-module, the output from **PLINK** can be linked using the OpenEdition `cc` command. The `-e //` option should be added the `cc` command to indicate that the entry-point is not the default Language Environment entry point expected by `cc`. The Systems/C runtime library will specify its own entry-point.

For example, to pre-link and link the object "`myfunc.o`" and produce the `HFS` load-module "`myprog`" under the OpenEdition shell (assuming `/usr/local/dignus` is the installation location), simply run **PLINK**:

```
plink -omyprog.o myfunc.o "-S/usr/local/dignus/objs_rent/&m"
```

then use the OpenEdition `cc` command:

```
cc -e // -omyprog myprog.o
```

to produce the `myprog` load-module. `myprog` can then be invoked as any other OpenEdition program.

For more information about creationg OpenEdition programs with Systems/C, see the *Systems/C C Library* manual.

Using PLINK to pre-link IBM C objects

As well as pre-linking Systems/C and Systems/C++ modules, **PLINK** can perform the function of the IBM pre-linker, `EDCPRLK`, for IBM C/C++ object modules.

PLINK automatically recognizes when the input contains IBM C/C++ object modules, and switches to "IBM pre-linking mode." It will then perform the same functions that the IBM pre-linker performs, using the same name-mapping algorithms, DLL processing and C++-specific processing.

If none of the primary input files, or any file they `INCLUDE`, is an IBM object, then the *–fc370* option must be manually specified if the IBM pre-linker processing is desired.

When pre-linking IBM objects, **PLINK** will properly process the IBM `IMPORT` and `RENAME` input cards. The *–fdllname* and *–fsysdefsd* options specify the resulting DLL name and name of the DLL definition file for any DLL exports that are encountered in the input objects.

When in IBM pre-linking mode, the PLINK-generated listing will appear similar to the IBM pre-linker listing.

When converting from a mainframe environment to a cross environment; any PDS libraries used for pre-linking should be converted to **DAR** archives on the cross-platform host. Care should be taken to ensure that AUTOCALL processing is consistent between the mainframe and cross-platform environments. If the PDS libraries have been processed by the IBM *Object Librarian*, then the equivalent **DAR** archive on the cross-platform host should be processed by the **DRANLIB** utility. When a PDS is processed by the IBM *Object Librarian*, a member named `@@DC370$` will be present. If that member is present, then the cross-platform **DAR** archive should be processed with the **DRANLIB** utility to ensure that proper "extended-name" AUTOCALLing will result. If the `@@DC370$` member is not in the PDS library, then **DRANLIB** should not be applied to the `DAR` archive, so that AUTOCALLing will use the member names.

If the resulting **DAR** archive is not processed by **DRANLIB**, then care should be taken to ensure that all of the names available in the PDS are also available in the **DAR** archive. This might include copying any PDS ALIASed members in the **DAR** archive so that all member names will be present. bf DAR archives have no "alias" facility, so a separate copy will be required.

During pre-linking, **PLINK** follows the autocall method of the IBM pre-linker `EDCPRLK`. That is, if the unresolved name is a "long name", autocall processing is not performed unless the specified search location is a **DAR** archive that has been processed with **DRANLIB**, or a PDS that has been processed with **DPDSLIB**. IBM's definition of a "long name" is one that is longer than 8 characters or has any lower-case letters.

If the *–b* or *–bonly* option is specified, indicating that the pre-linked object deck should then be "bound", any subsequent autocall processing that may occur follows the normal **PLINK** rules and not the rules used by `EDCPRLK`.

Using PLINK to directly create load modules

PLINK can, on cross-platform hosts, optionally perform the final linking, or *binding* step and produce a load module. The load module will reside in a **PLINK**-created TSO TRANSMIT file suitable for receiving the resulting load-module with the TSO RECEIVE command.

When the *–b* or *–bonly* options are specified, **PLINK** will internally perform the final linking steps, or *binding*, and internally build the resulting load module. The *–b* option causes **PLINK** to perform its normal pre-linking steps, following by the binding step to create the load module. The *–bonly* option causes **PLINK** to only perform the binding process, skipping the normal **PLINK** pre-linking processing.

When binding is performed, the **PLINK** listing output will contain a load module map and symbol cross reference sections that are similar to those produced by the IBM linker.

After internally binding the load module, **PLINK** will create a TSO TRANSMIT file, which describes the resulting load module as if it resided on a 3390 direct access storage device. If the specified output file already exists and is a TSO TRANSMIT file, **PLINK** will update it, preserving any other members already present.

The resulting TSO TRANSMIT file should then be copied to a fixed-block data set, with a logical record length of 80 bytes. Then, the RECEIVE command can be employed to realize the load module on an MVS, OS/390 or z/OS system.

The *–syslmod* option defines the name of the PDS on the mainframe platform. *–syslmod* can specify a PDS name and a member name. If a member name is not specified, **PLINK** will use a name specified on a `NAME` card in the input, or the name `TEMPNAM0`.

The *–blksize* option can be used to define the block size of the resulting PDS file. If no *–blksize* option is specified, **PLINK** uses a value of 27998, the optimum block size for a 3390 device.

For example, the following **PLINK** command will pre-link `obj1.obj`, `obj2.obj` and `obj3.obj`, producing the output TSO transmit file `prog.xmi`. The final location for the resulting load module will be in the PDS named "`MY.PROGS.LOAD`" in the member named "`PROG`".

```
plink -o prog.xmi -b -syslmod="MY.PROGS.LOAD(PROG)" obj1.obj
        obj2.obj obj3.obj
```

The `prog.xmi` file can then be copied to an FB-80 data set, and the RECEIVE command will produce the `MY.PROGS.LOAD(PROG)` load-module.

For more information about using the RECEIVE command, see the IBM publication *TSO/E Command Reference.*

Using PLINK to link programs on OS/390

Before execution, programs must be prepared, optionally using the Systems/C pre-linker, **PLINK**, then the IBM BINDER.

Systems/C provides two versions of the Systems/C C libraries, one for RENT programs and one for non-RENT programs. If you are using the Systems/C library, it is important to link with the appropriate version. If any source programs reference variables found in the Systems/C library (e.g. `errno`) and that program was compiled with the *–frent* option, then the re-entrant version of the Systems/C library should be used. Using the incorrect version of the library will cause strange run-time errors. The installation instructions for your particular host platform will detail where to find the correct Systems/C library. Normally the Systems/C library is specified as the last library to use for AUTOCALL resolution in the **PLINK** step. Furthermore, **PLINK** must be used for re-entrant programs that use the Systems/C library or to take advantage of **DAR** archive libraries for external reference resolution.

In the following example JCL, there are three objects to link together to form the resulting executable, *MAIN*, *SUB1*, and *SUB2*, representing a main module and two supporting sub-modules. These are found in the PDS `MY.PDS.OBJ`. The resulting executable is written to `MY.PDS.LOAD(MPROG)`.

```
//LINK JOB
//PLINK EXEC PGM=PLINK,REGION=2048K
//STEPLIB DD DSN=Systems/C load library,DISP=SHR
//STDOUT  DD SYSOUT=*
//STDERR  DD SYSOUT=*
//SYSLIB  DD DSN=DIGNUS.LIBCR,DISP=SHR
//SYSMOD  DD DSN=&&PLKDD,UNIT=VIO,DISP=(NEW,PASS),
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//INDD    DD DSN=MYPDS.OBJ,DISP=SHR
//SYSIN   DD *
 INCLUDE INDD(MAIN)
 INCLUDE INDD(SUB1)
 INCLUDE INDD(SUB2)
//STDIN   DD *
//LINK EXEC PGM=IEWL,REGION=2M,PARM=('LIST',
//  'MAP,XREF,LET',
//  'ALIASES=NO,UPCASE=NO,MSGLEVEL=4,EDIT=YES')
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
```

First **PLINK** is invoked, specifying the inclusion of the three object modules and the Systems/C C reentrant library. This step could have been performed on a cross-platform host, running **PLINK** there. Then the IBM BINDER is invoked for final linking and generation of the resulting load module.

Systems/C shared libraries

The Systems/C runtime supports UNIX-style shared libraries, allowing programs to be divided into logical load-modules which are dynamically loaded at runtime. Each such load module is called a "shared library." Code that uses a shared library or is in a shared library is compiled with the *–fpic* option to **DCC** or **DCXX**.

Shared library files

A "shared library" consists of two items, the SO object and the shared library module.

The shared library module is a self contained load module, linked on the host to produce a loadable module. It contains all of the definitions of code and data that make up the shared library.

The SO object is an object deck which is linked into the program that uses the shared library module. It contains the function stubs and "`&var`" initializers for all the defined items in a shared library. The SO object also contains a new card, the `.SO` card. This card communicates to PLINK the runtime name of the shared library module.

When pre-linking, should PLINK discover a `.SO` card, it adds the named module to the list of modules. PLINK then appends an `@@SOLST#` CSECT to the generated output. The `@@SOLST#` contains the names of all of the shared modules which will be needed at runtime for this executable.

Data references

The compiler generates references to locally-defined data (and static data) as normal.

For undefined (extern) reentrant data, the compiler generates a special Q-con with an "`&`" prefix, such as "`&var`". The corresponding location within the PRV will be filled in with the address of the variable, rather than with the variable itself. This extra level of indirection allows a PRV within one library to contain a reference to a symbol defined in another library that has its own PRV.

Data definitions

Defined data is generated as normal.

For externally-visible defined data, an extra definition is created for the "`&var`" Q-con, with a special reentrant initializer script to indicate that it should be initialized with the address of the dynamic variable "`var`".

Thus, only one initializer will be executed per module, saving runtime startup cost. Furthermore, references to extern RENT data only suffer one additional instruction when computing their address (note that the compiler can "cache" this address and doesn't need to frequently recompute it in the same basic block).

Functions

Remote function pointers are used that point to a two-word unit containing both the PRV and the address of the start of the function's code.

For calls directly to a named function, the compiler generates the call as normal. If the reference is satisfied by an SO object, then the function refers to a **PLINK**-generated stub routine. This stub is responsible for determining the runtime address of the function along with the shared-library's allocated PRV (there are runtime initializers for these functions). The stub then saves the current PRV, sets the new one, branches to the function, restores the PRV and returns to the caller. Essentially, the stub-function converts a local function call into a remote one.

A significant advantage of this approach is that calls to functions within the same module suffer no added cost. Only calling outside the module results in any dynamic linkage overhead.

Also note that remote function pointers can be successfully passed across shared-library modules.

Runtime support

On startup, the runtime loads any shared library modules specified in the `@@SOLST#` CSECT (if present). It creates a dictionary of the symbols provided by each library that it loads. If a library is loaded that has dependencies on another library, the other library is loaded to satisfy the dependency.

Then it allocates the PRVs for all of the loaded libraries, and executes the reentrant initialization scripts for them. The reentrant initialization scripts contain two new entries, one which resolves to the PRV for a named library and another which resolves to the address of a shared symbol. They are resolved using the shared library dictionaries.

Shared libraries may also be loaded using `dlopen()`. See the Systems/C C Library manual for more information.

Building a shared library

Each shared library has a name, provided by the **PLINK** option *–soname=name.* It should correspond with the name of the shared library load module, as it will be used to find that load module.

To build the shared library module and its associated SO object, a two-step process is used.

Step #1

First, **PLINK** must be executed with the *–shared* and *–asm* options, and with all of the objects within the shared library specified on the command line. This will cause the files `soref.asm` and `sodef.asm` to be generated. Alternative filenames for these two can be specified with the *–soref=soref.asm* and *–sodef=sodef.asm* options.

`sodef.asm` contains stubs for all of the symbols defined within this shared library and exported for use by other load modules. After it is assembled with **DASM**, it will provide the SO object.

If you do not use *–exclude_allrefs*, then `sodef.asm` also includes external references for any undefined symbols within the shared library, in hopes of later AUTOCALL resolution. That is the historical behavior, but if the AUTOCALL is not successful then it will make it harder to track down the source of these undefined symbol references. So it is now recommended to use *–exclude_allrefs* any time you use *–shared –asm*.

`soref.asm` contains stubs for the undefined symbols within this shared library (i.e., references to symbols outside of this library). It must be assembled with **DASM** and the resulting object deck should be passed as an argument to **PLINK** for the next step.

Both `soref.asm` and `sodef.asm` use a macro called `DCCLSTUB`, which provides the code for a stub routine. It needs to manage setting and resetting the PRV across shared library function calls. `DCCLSTUB` may be replaced with user-supplied code.

Step #2

Then **PLINK** is run again, this time with just *–shared*, and the list of all of the objects in the shared library, including the one produced from `soref.asm`. The resulting object is the shared library module. It must be turned into a load module by either final linking on the mainframe or using the *–b* option to **PLINK**.

Example

In the following example, we compile the file `a.c` and assemble it to produce `a.obj`, then we build the shared-library `A`.

Compiling:

```
dcc -fpic -o a.asm a.c
dasm -Lmaclib -macext . -oa.obj a.asm
```

Pre-linking, step #1:

```
plink -shared -asm -soname A -sodef=adef.asm -soref=aref.asm a.obj
dasm -Lmaclib -macext . -oaref.obj aref.asm
dasm -Lmaclib -macext . -oadef.obj adef.asm
```

At this point `adef.obj` is the SO object for shared library `A`, and should be linked in with any program that uses shared library `A`.

Pre-linking, step #2 (build the shared library module):

```
plink -oaload.obj -shared -soname A a.obj aref.obj
```

At this point, `aload.obj` is the pre-linked object deck that should be linked into the load module A on the host platform. Since `aref.obj` has now been used, it is safe to delete `aref.asm` and `aref.obj`.

To continue with this example, let's compile `myprog.c` which uses the shared library `A`.

First compile and assemble `myprog.c` (note that we specify the *–fpic* option):

```
dcc -omyprog.asm -frent -fpic myprog.c
dasm -Lmaclib -macext . -omyprog.obj myprog.asm
```

Now simply pre-link `myprog.obj` with the the SO object file for `A` (`adef.obj`), along with the Systems/C library, creating `myload.obj`:

```
plink -omyload.obj myprog.obj adef.obj "-Sobjs_rent/&m"
```

`myload.obj` can be linked on the host platform. At runtime, the program will attempt to load the shared-library-module named `A`, because that was the name specified in the *–soname* option on the PLINK steps.

DPDSLIB

Using DPDSLIB

DPDSLIB is a utility provided on OS/390 and z/OS which allows autocall resolution of PDS members to account for long symbol names within the object members. Typically, when resolving names based on PDS members, **PLINK** attempts to resolve the reference by opening the PDS member with the same filename as the reference. This limits the length of external symbols to 8 characters. Also, **PLINK** converts the reference name to upper-case letters to find the PDS member, which may cause conflicts.

DPDSLIB will examine all of the objects in a PDS, creating a table-of-contents PDS member that describes each member in the PDS and the symbols it defines. It will then add this table-of-contents file to the PDS for **PLINK** to use later. Files in the PDS which are not GOFF, XSD or ESD format will be ignored. The name of the dictionary file in the PDS is `##SYMDEF`. In this regard, **DPDSLIB** performs a similar function to a PDS library as **DRANLIB** performs to a **DAR** archive.

Note that if members are added to, or changed in the PDS, the **DPDSLIB** program should be executed again to update the table-of-contents file.

Running DPDSLIB

DPDSLIB can be found in the Systems/C executable load module PDS for your installation, along with the other Systems/C programs. **DPDSLIB** may be executed either in TSO or with JCL as a batch program.

DPDSLIB has only one option, the name of the PDS to examine.

DPDSLIB examples

For example, if the PDS object library is named `USER.MY.LIB`, the following command under TSO would examine each of the members in `USER.MY.LIB` and create the `##SYMDEF` table-of-contents file.

```
      DPDSLIB //DSN:USER.MY.LIB
```

Note that the specification of the PDS uses the Systems/C file naming conventions.

The following example demonstrates how to use **DPDSLIB** in a batch environment. In this example, the name of the PDS library to examine is given as the `PARM` value on the `DPDSLIB EXEC` statement.

```
      ...
      //*
      //* Execute DPDSLIB to add the table-of-contents
      //* file to the PDS named in
      //* the DD LIBRARY.
      //*
      //DPDSLIB EXEC PGM=DPDSLIB,
      //             PARM='LIBRARY'
      //STEPLIB DD DSN=Systems/C load library,DISP=SHR
      //STDOUT  DD SYSOUT=*
      //STDERR  DD SYSOUT=*
      //LIBRARY DD DSN=PDS library name,DISP=SHR
```

DPDSLIB also supports multiple `DSN`s concatenated into a single `DD` definition. In this case, **DPDSLIB** will treat each `DSN` as its own library, as if **DPDSLIB** were executed on each `DSN`. For example, in the following JCL, the PDSs `MY.PDS1`, `MY.PDS2` and `MY.PDS3` will each be examined in turn, and a `##SYMDEF` table-of-contents member will be added to each.

```
      ...
      //*
      //* Execute DPDSLIB to add the table-of-contents
      //* file to the PDS named in
      //* the DD LIBRARY.  In this example the DD
      //* refers to several DSNs
      //*
      //DPDSLIB EXEC PGM=DPDSLIB,
      //             PARM='LIBRARY'
      //STEPLIB DD DSN=Systems/C load library,DISP=SHR
      //STDOUT  DD SYSOUT=*
      //STDERR  DD SYSOUT=*
      //LIBRARY DD DSN=MY.PDS1,DISP=SHR
      //        DD DSN=MY.PDS2,DISP=SHR
      //        DD DSN=MY.PDS3,DISP=SHR
```

DAR

Using DAR

The Systems/C archive utility, **DAR**, creates and maintains groups of files combined into an archive. Once an archive has been created, new files can be added and existing files can be extracted, deleted or replaced.

Files are named in the archive by a single component, i.e., if a file referenced by a path containing a slash (`/`) is archived on a UNIX platform, it will be named by the last component of that path. When matching paths listed on the command line against file names stored in the archive, only the last component of the path will be compared.

All informational and error messages use the path listed on the command line, if any was specified; otherwise the name in the archive is used. If multiple files in the archive have the same name, and paths are listed on the command line to "select" archive files for operation, only the *first* file with a matching name will be selected.

The normal use of **DAR** is for creation and maintenance of libraries suitable for use with the Systems/C pre-linker, **PLINK**, although it is not restricted to this purpose.

Running DAR

On UNIX and Windows cross-platform hosts, the **DAR** utility is located in the Systems/C installation directory, as `dar` on UNIX, or `DAR.EXE` for Windows. On OS/390, **DAR** is found in the Systems/C installation PDS, as the `DAR` member.

The **DAR** utility exits with a zero (0) return code on success, and greater-than zero (0) if an error occurs.

During processing, DAR requires several temporary files. On UNIX hosts, these will reside in `/tmp/dar.XXXXX`. On Windows hosts, these will be named `DAR#####.TMP` and will be located in the current directory. On OS/390, the files `SYSUT1-SYSUT5` may be employed (depending on the operation requested) and must be properly allocated.

The **DAR** utility accepts the following possible command lines formats:

> **DAR** -d [-Tv] *archive file* ...
> **DAR** -m [-Tv] *archive file* ...
> **DAR** -m [-abiTv] *position archive file* ...
> **DAR** -p [-Tv] *archive* [*file* ...]
> **DAR** -q [-cTv] *archive file* ...
> **DAR** -r [-cuTv] *archive file* ...
> **DAR** -r [-abciuTv] *position archive file* ...
> **DAR** -t [-Tv] *archive* [*file* ...]
> **DAR** -x [-ouTv] *archive* [*file* ...]

The *archive* file name is provided after any options (possibly preceded by a *position* option, which names a member in the archive.) Following the archive file name is a list of member names, or *file*s within the archive.

Also on Windows and OS/390 host platforms, **DAR** accepts a –@*filename* option. This option can be used to specify a command line larger than the host operating system supports. The contents of *filename* are examined and inserted in the command line argument list where the option appears.

DAR options

DAR options:

–a   A positioning modifier used with the options –*r* and –*m*. The files are entered or moved *after* the archive member *position*, which must be specified.

–b   A positioning modifier used with the options –*r* and –*m*. The files are entered or moved *before* the archive member *position*, which must be specified.

–c   Whenever an archive is created, an informational message to that effect is written to standard error. If the –*c* option is specified, **DAR** creates the archive silently.

–d   Delete the specified archive files.

–i   Identical to the –*b* option.

–m   Move the specified archive files within the archive. If one of the options –*a*, –*b* or –*i* is specified, the files are move before or after the *position* file in the archive. If none of these options are specified, the files are moved to the end of the archive.

–o   Set the access and modification times of extracted files to the modification time of the file when it was entered into the archive. This will fail if the user does not have sufficient authority for the operation, or the host file system does not support it.

–p   Write the contents of the specified files to the standard output. If no files are specified, the contents of all the files in the archive are written in the order they appear in the archive.

–q   (Quickly) append the specified tiles to the archive. If the archive does not exist a new archive file is created. This option can be must faster than the –r option, when creating a large archive piece-by-piece, as no checking is done to see if the files already exist in the archive.

–r   Replace or add the specified files to the archive. If the archive does not exist a new archive file is created. Files that replace existing files do not change the order of the files within the archive. New files are appended to the archive unless one of the options –a –b or –i is specified.

–T   Select and/or name archive members using only the first fifteen characters of the archive member or command line file name. The historic archive format had sixteen bytes for the name, but some historic archiver and loader implementations were unable to handle names that used the entire space. This means that file names that are not unique in their first fifteen characters can subsequently be confused. A warning message is printed to the standard error output if any file names are truncated.

–t   List the specified files in the order in which they appear in the archive, each on a separate line. If no files are specified, all files in the archive are listed.

–u   Update files. When used with the –r option, files in the archive will be replaced only if the disk file has a newer modification time than the file in the archive. When used with the –x option, files in the archive will be extracted only if the archive file has a newer modification time than the file on the disk. On those hosts file systems that don't support a modification time in the file system (i.e. OS/390 PDS members) this option produces an error message on the standard error output.

–v   Provide verbose output. When used with the –d, –m, –q or –x options, **DAR** gives a file-by-file description of the archive members. This description consists of three, white-space separated fields: the option latter, a dash ('-') and a file name. When used with the –r option, **DAR** displays the description as above but the initial letter is an "a" if the file is added to the archive and an "r" if the file replaces a file already in the archive.

        When used with the –p option, the name of each printed file, enclosed in less-than (<) and greater-than (>) characters, is written to the standard output before the contents of the file; it is preceded by a single newline character, and followed by two newline characters.

When used with the *–t* option, **DAR** displays listing of information about the members of the archive. This listing is similar to the UNIX "`ls -l`" command, and consists of eight, white-space separated fields: the file permissions, the decimal user and group identifier numbers separated by a single slash (`/`), the file size (in bytes), the file modification time (in the date format `"%b %e %H:%M %Y"`), and the name of the file.

–x   Extract the specified archive members into the files named by the command line arguments. If no members are specified, all the members of the archive are extracted into the current directory.

If the file does not exist, and the host file system supports it, it is created; if it does exist, the owner and group will be unchanged. The file access and modification times are the time of the extraction (but see the *–o* option for alternatives.) The file permissions will be set to those of the file when it was entered into the archive; this will fail if the user does not have sufficient authority for the operation, or the host file system does not support it.

DAR examples

The following JCL is an example of how to use DRANLIB on OS/390 or z/OS. Note the definition of the temporary file `SYSUT0`. This example creates the archive referenced in the `ARCHIVE` DD statement, using the three members `MEM1`, `MEM2` and `MEM3` from the `OBJS` DD.

```
...
//*
//* Execute DAR to create the archive named in
//* the DD ARCHIVE.
//*
//DAR EXEC PGM=DAR,REGION=2049K,
//           PARM='-@PARMS'
//STEPLIB DD DSN=Systems/C load library,DISP=SHR
//PARMS   DD *
-rv,ARCHIVE,OBJS(MEM1),OBJS(MEM2),OBJS(MEM3)
//STDOUT  DD SYSOUT=*
//STDERR  DD SYSOUT=*
//SYSUT0  DD UNIT=SYSDA,SPACE=(CYL,(5,1)),
//          BLKSIZE=80,LRECL=1,RECFM=FB
//ARCHIVE DD DSN=archive name,
//          DISP=(NEW,CATLG,DELETE),
//          SPACE=(CYL,(10,2),RLSE),
//          BLKSIZE=80,LRECL=1,RECFM=FB
//OBJS    DD DSN= objs PDS,DISP=SHR
```

On UNIX or Widows platforms, the DAR program is invoked in a manner similar to the typical UNIX `ar` program. This example creates the archive named `libstuff.a`, from the three objects in the `objs` directory on a UNIX platform:

```
dar rv libstuff.a objs/mem1 objs/mem2 objs/mem3
```

DRANLIB

Using DRANLIB

The **DRANLIB** utility is used to create a table-of-contents member for **DAR** archive libraries. The **PLINK** pre-linker uses this table-of-contents member to resolve external references. This table is named "`__.SYMDEF`" and is prepended to the archive. Files in the archive which are not GOFF, OBJ or XOBJ format object decks and symbols which are uninteresting to **PLINK** are ignored.

Running DRANLIB

On UNIX platforms, the **DRANLIB** utility is found in the Systems/C installation directory as the file `dranlib`. On Windows hosts, it is named `DRANLIB.EXE`. On OS/390 and z/OS, the **DRANLIB** utility is found in the Systems/C load module PDS as the `DRANLIB` member.

DRANLIB has the following command line:

```
DRANLIB [-t] file ...
```

where any number of **DAR** archive *file*s may be specified.

DRANLIB requires temporary files to perform this operation. On UNIX platforms, these are found in `/tmp/dranlib.XXXXX`. On Windows platforms, the files `DRL#####.TMP` in the current directory will be used. On OS/390, the files `SYSUT1-SYSUT3` will be used and should be appropriately allocated.

DRANLIB returns with a return code of 0 on success, any non-zero return code indicates errors.

DRANLIB options

–t   Set the modification time of the generated ⎽⎽.SYMDEF member in the archive. Some linkers (but not the Systems/C pre-linker **PLINK**) compare this time with the modification time of the archive to verify that the table is up-to-date with respect to the archive. If the modification time has been changed without any change to the archive (for example, by copying the archive), this option can be used to "touch" the modification time so that it appears the table is up-to-date. This is also useful on UNIX hosts after using the –t option of the make command. Note that this option is ineffective when running **DRANLIB** on OS/390 and z/OS.

DRANLIB examples

Some typical JCL for using DRANLIB on OS/390 might be:

```
//*
//* Execute DRANLIB to add the definition
//* symbol table to the archive named in
//* the DD ARCHIVE.
//*
//DRANLIB EXEC PGM=DRANLIB,REGION=2049K,
//             PARM='ARCHIVE'
//STEPLIB DD DSN=Systems/C load library,DISP=SHR
//STDOUT  DD SYSOUT=*
//STDERR  DD SYSOUT=*
//SYSUT0  DD UNIT=SYSDA,SPACE=(CYL,(5,1)),
//          BLKSIZE=80,LRECL=1,RECFM=FB
//SYSUT1  DD UNIT=SYSDA,SPACE=(CYL,(5,1)),
//          BLKSIZE=80,LRECL=1,RECFM=FB
//ARCHIVE DD DSN=archive name,DISP=(MOD)
```

On UNIX and Windows, the typical command for using DRANLIB is:

     dranlib   *archive file name*

For example, to add a symbol definition file to the archive myarchive.a, the command would be:

     dranlib myarchive.a

GOFF2XSD

Using GOFF2XSD

GOFF2XSD is a program supplied with Systems/C which converts GOFF format object code to XSD format object code. The IBM HLASM generates GOFF format object code when the XOBJECT option is enabled. GOFF format, as well as XSD format, allows for identifiers which are longer than the 8-character limit imposed by the older ESD format. The Systems/C pre-linker **PLINK** can directly process either GOFF or XSD object format files, converting them to ESD format which may be required by some older linkers.

Because **PLINK** can accept GOFF input directly, this utility is now no longer required for **PLINK** and should be considered deprecated.

Typically, **GOFF2XSD** is used on an OS/390 or z/OS host, as part of the assembly step, but it is available and can be used on any Systems/C supported platform.

Running GOFF2XSD

GOFF2XSD accepts a *–o filename* option followed by the name of the input file as parameters. The *–o filename* option specifies where the output should be written. On OS/390 and z/OS, if *–o filename* is not specified, the output is written to the `SYSOUT` DD. Also, on OS/390, the output data set should be a fixed block file with 80 byte records (`RECFM=FB,LRECL=80.`)

On OS/390 and z/OS, if no input file is specified, **GOFF2XSD** reads from the `SYSIN` DD.

For example, if the HLASM step was named `ASM`, the **GOFF2XSD** `SYSIN` DD could refer back to the output of HLASM, and generate a `SYSOUT` object module suitable for inclusion by **PLINK** with the following JCL:

```
...
//*
//* Execute GOFF2XSD which translates the
//* HLASM-produced GOFF object into an
//* XSD-format object.
//*
//GOFF2XSD EXEC PGM=GOFF2XSD,REGION=2049K
//STEPLIB  DD DSN=Systems/C load library,DISP=SHR
//STDOUT   DD SYSOUT=*
//STDIN    DD SYSOUT=*
//SYSIN    DD DSN=*.ASM.SYSLIN,DISP=(OLD,DELETE)
//SYSOUT   DD DSN=&&OBJ,UNIT=VIO,DISP=(NEW,PASS),
//      SPACE=(32000,(30,30)),
//      DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
```

Note that the `STDIN` and `STDOUT` DDs are defined as well, **GOFF2XSD** writes any informative or error messages to those data sets.

On a cross-platform host, the following will translate the file named `goffin`, creating the XSD object file `xsdout`:

```
goff2xsd -o xsdout goffin
```

DCCPC

Using DCCPC

The Dignus CICS Command Processor, **DCCPC**, is used to translate C source code which uses `EXEC CICS` commands into normal C code prior to invoking **DCC**. This is especially useful in cross environments where IBM's translators cannot be used. For further information about CICS, see the IBM publication *CICS Transaction Server for z/OS: CICS Application Programming Reference* document number SC34-5994-02.

DCCPC implements all of commands from **CICS/ESA V4R1**, and many of the newer **CICS TS** commands as well.

Running DCCPC

On Windows and Unix, **DCCPC** is executed with a command line of the form

```
dccpc [options] [input file]
```

If no input file is specified, input is read from **stdin**.

When run on OS/390 and z/OS, options may be specified in the `PARM` statement. If no input file is specified, the `SYSIN` DD is used. Output defaults to `SYSPUNCH`, which would typically be used as input for **DCC** in the next step. Informational and error messages are output to `STDOUT` and `STDERR` DDs. The following JCL could be used on OS/390:

```
//DCCPC    JOB
//DCCPC    EXEC PGM=DCCPC,PARM='options'
//STEPLIB  DD DSN=Systems/C load library,DISP=SHR
//STDERR   DD SYSOUT=*
//STDOUT   DD SYSOUT=*
//SYSPUNCH DD SYSOUT=*
//SYSIN    DD *
     <C source>
```

DCCPC Options

Options for **DCCPC** are summarized in the table below.

–A	process assembly source
–C	process C source
–o *file*	place translated output in the file named *file*
–fdli / –fnodli	enable/disable support for `EXEC DLI`
–fgds / –fnogds	enable/disable support for GDS commands
–fsp / –fnosp	enable/disable support for System Programmer commands
–fcols=*n*	the output in C mode will be *n* columns
–fseq	the output in C mode will include sequence numbers
–fmrc / –fnomrc	enable/disable use of mainframe-style return code
–fflag=*code*	output only error messages of a certain priority
–fepilog / –fnoepilog	enable/disable use of `DFHEIRET` macro
–fprolog / –fnoprolog	enable/disable use of `DFHEISTG`, `DFHEIEND`, and `DFHEIENT` macros
–ferrlist / –fnoerrlist	enable/disable listing errors on *stderr*
–fcpp / –fnocpp	enable/disable C++ mode
–fexci / –fnoexci	enable/disable EXCI mode
–fleasm / –fnoleasm	enable/disable LE ASM compatibility
–fvse / –fnovse	enable/disable VSE compatibility
–fedf / –fnoedf	enable/disable EDF mode
–v	print version information

The –A option (process assembly source)

The *–A* option tells **DCCPC** to process assembly source code.

The –C option (process C source)

The *–C* option tells **DCCPC** to process C source code instead of assembly code. *–C* is the default for **DCCPC**.

The –o *file* option (specify the name of the output file)

The *–o file* option specifies that the translated output should go to a file other than
the default. On Windows and Unix systems, the default is `ccp.out`, while in OS/390
it is the `SYSPUNCH` DD.

The –fdli and –fnodli options (enable/disable `EXEC DLI`)

The *–fdli* option enables support for `EXEC DLI` statements as well as `EXEC CICS`
statements.

The –fgds and –fnogds options (enable/disable GDS commands)

The *–fgds* option enables support for GDS commands (commands of the form `EXEC`
`CICS GDS ...`). The default is *–fnogds*.

**The –fsp and –fnosp options (enable/disable System Programmer com-
mands)**

The *–fsp* option enables support for System Programmer commands. The default
is *–fnosp*.

The –fcols=n option (specify column width)

When processing C source code the *–fcols=n* option instructs **DCCPC** to use only
n columns in its output. Assembly code is always limited to 72 columns with
continuations.

The –fseq option (generate sequence numbers)

When processing C source code the *–fseq* option causes **DCCPC** to generate se-
quence numbers in the output. This implicitly sets `-fcols=72`. The sequence num-
bers appear in columns 73-80.

**The –fmrc and –fnomrc options (enable/disable mainframe-style return
codes)**

The *–fmrc* option specifies that the translator should use mainframe-style return
codes to indicate the exact error level reached. The *–fnomrc* option specifies that

the translator should use Unix-style return codes that are either 0 (success) or 1 (errors). On OS/390 *–fmrc* is the default, while on cross-platform hosts *–fnomrc* is the default.

The –fflag=*code* option (output only error messages of a certain priority)

The *–fflag=code* option specifies that only error messages at least as severe than *code* should be displayed. Valid values for *code* are `I` for informational messages, `W` for warnings, `E` for errors, and `S` for severe errors. The default is *–fflag=I*.

The –fepilog and –fnoepilog options (enable/disable use of `DFHEIRET` macro)

The *–fnoepilog* option specifies that the `DFHEIRET` macro should not be invoked in the translated assembly source, This option is needed to make the CICS `RETURN` command effective. The default is *–fepilog*.

The –fprolog and –fnoprolog options (enable/disable use of `DFHEISTG`, `DFHEIEND` and `DFHEIENT` macros)

The *–fnoprolog* option specifies that the `DFHEISTG`, `DFHEIEND` and `DFHEIENT` macros should not be invoked in the translated assembly source. These macros define local storage that is allocated a program start up. The default is *–fprolog*.

The –ferrlist and –fnoerrlist options (enable/disable listing of errors on *stderr*)

The *–ferrlist* option enables the listing of errors on *stderr*. The *–fnoerrlist* option specifies that errors should just be listed as comments in the translated output file and is the default.

The –fcpp and –fnocpp options (enable/disable C++ mode)

The *–fcpp* option can be used with the *–C* option to use C++ instead of C for input and output. *–fcpp* has no effect if *–C* is not specified.

The –fexci and –fnoexci options (enable/disable EXCI mode)

The *–fexci* option instructs **DCCPC** to run in EXternal Call Interface mode, for processing files which contain only a special form of the `EXEC CICS LINK` command. When run in EXCI mode, no other commands will be translated. Outside of EXCI mode, this special form of the `EXEC CICS LINK` command is unavailable.

The –fleasm and –fnoleasm options (enable/disable LE ASM compatibility)

The *–fleasm* option changes some macro parameters to create a Language Environment conforming assembler program, rather than one to be loaded in the CICS environment. It should only be used in assembler mode (*–A*). The default behavior is *–fnoleasm*, to generate a program to be executed from the CICS environment.

The –fvse and –fnovse options (enable/disable VSE compatibility

By default, **DCCPC** generates output compatible with the z/OS or MVS versions of the IBM CICS preprocessor. However, the CICS preprocessor for VSE supports a few new commands (`SPOOLCLOSE REPORT`, `SPOOLOPEN ESCAPE`, `SPOOLOPEN MAPNAME`, `SPOOLOPEN REPORT`, `SPOOLOPEN RESUME`, `SPOOLWRITE MAPNAME`, and `SPOOLWRITE REPO`), and has alternative translations for a few others (`INQUIRE PROGRAM` and `INQUIRE TASK`). When *–fvse* is supplied on the commandline, **DCCPC** will generate translations compatible with the VSE preprocessor.

The –fedf and –fnoedf options (enable/disable EDF mode)

The default is *–fedf*. If *–fnoedf* is specified then the `X'40'` bit in the common flags area of each `DFHECALL` invocation is set.

The –v option

The *–v* option causes **DCCPC** to produce a banner indicating the version and copyright information, then to exit with a return code of 0.

DB2PPC

Using DB2PPC

The Dignus DB2 Command Processor, **DB2PPC**, is used to translate C source code which uses `EXEC SQL` commands into normal C code and a DBRM file prior to invoking **DCC**. This is especially useful in cross environments where IBM's translators cannot be used. For further information about DB2, see the IBM publication *DB2 UDB for z/OS V8 SQL Reference* document number SC18-7426-03.

Running DB2PPC

On Windows and Unix, **DB2PPC** is executed with a command line of the form

      db2ppc *[options] [input file]*

If no input file is specified, input is read from **stdin** and C output is generated on **stdout** and DBRM output is generated in `out.dbrm`.

If an input file is specified but no output files are specified then the output file names are based off of the input file name. For example, if the input is `file.c`, then the translated C output will be in `file_pp.c` and the DBRM will be in `file.dbrm`.

When run on OS/390 and z/OS, options may be specified in the `PARM` statement. If no input file is specified, the `SYSIN` DD is used. C output defaults to `SYSCIN`, which would typically be used as input for **DCC** in the next step. DBRM output defaults to the `DBRMLIB` DD. Informational and error messages are output to `STDOUT` and `STDERR` DDs. The following JCL could be used on OS/390:

```
//DB2PPC    JOB
//DB2PPC    EXEC PGM=DB2PPC,PARM='options'
//STEPLIB   DD DSN=Systems/C load library,DISP=SHR
//STDERR    DD SYSOUT=*
//STDOUT    DD SYSOUT=*
//SYSCIN    DD C output dataset
//DBRMLIB   DD DBRM output dataset
//SYSIN     DD *
      <C source>
```

Defining DB2PP global variables

The DB2 Command Processor produces definitions for some global variables, such as `SQLTEMP`. When compiled with the IBM C compiler, it defines these global variables redundantly in each compilation unit, relying on "common mode linking" to clean up the duplicate definitions. Systems/C and Systems/C++ are designed to conform with the "one definition rule" instead, so by default these global variables are declared `extern`, instead of defined. So one compilation unit must provide the single canonical definition, using the special `#define`, `__DB2PP_DEFINE_SYMS`. A common approach would be to add *—D__DB2PP_DEFINE_SYMS=1* to the **DCC** command-line for one of the compilation units that has been processed with **DB2PPC**.

DB2PPC Options

Options for **DB2PPC** are summarized in the table below.

–A / –C	process assembly / C source
–d *file*	place DBRM output in the file named *file*
–I *path*	search for include files in *path*
–o *file*	place translated output in the file named *file*
–macext *ext*	specify file extension to try for `EXEC SQL INCLUDE` statements
–flowerinc / –fnolowerinc	enable/disable lowercasing of include filenames
–fmrc / –fnomrc	enable/disable use of mainframe-style return code
–fnewfun / –fnonewfun	enable/disable support for V8 **NEWFUN** mode
–fattach=*mode*	specify attachment mode
–fconnect=1 / –fconnect=2	specify type for `CONNECT` statement
–fdate=*format*	specify a date format to override the system default
–ffloat=ieee / –ffloat=s390	specify type of floating point variables
–fpadntstr / –fnopadntstr	enable/disable padding of NUL-terminated strings
–fsql=all / –fsql=db2	specify which SQL dialect to accept
–fstdsql	use SQL rules for statements
–fversion=*ver*	set the version field in the **DBRM** header
–fcpp	enable C++ mode
–fonepass / –ftwopass	select one- or two-pass preprocessor operation
–ffold / –fnofold	enable/disable folding identifiers to uppercase
–fprog=*name*	override the DBRM "program name" header field
–fuser=*name*	override the DBRM "user name" header field
–ftimestamp=*hex*	override the DBRM "timestamp" header field
–v	print version information
–fdb2ver=v*N*	set DB2 compatibility version
–fccsid=*num*	set CCSID

The −A and −C options (process assembly / C source)

The *−A* option tells **DB2PPC** to process assembly source code. It implies *−ffold* and *−ftwopass* options, so if you wish to override these options then you must specify (i.e.) *−fonepass* at a point after *−A* on the command line.

The *−C* option tells **DB2PPC** to process C source code. The default is to process C source code.

The −d *file* option (place DBRM output in the file named *file*)

When *−d file* is specified, **DB2PPC** generates DBRM output in the specified file. If *−d* is not specified, then the output filename is determined from the input filename with the extension replaced by `.dbrm`. If no input filename is specified then output is placed in `out.dbrm` or the `DBRMLIB` DD.

The −I *path* option (search for include files in *path*)

Any paths specified with *−I path* will be searched (in the order given) when resolving `EXEC SQL INCLUDE` statements. The current directory ("`.`") is always searched.

If you wish to include a file with an extension, please enclose the file name in quotes, as in:

```
EXEC SQL INCLUDE "file.c"
```

See also *−macext*, which specifies other ways to deal with file extensions with `EXEC SQL INCLUDE`.

The −o *file* option (place translated output in the file named *file*)

When *−o file* is specified, **DB2PPC** generates translated C output in the specified file. If *−o* is not specified, then the output filename is determined from the input filename with the extension replaced by `_pp.c`. If no input filename is specified then output is on **stdout** or the `SYSCIN` DD.

The −macext *ext* option (specify file extension to try for `EXEC SQL INCLUDE` statements)

The *−macext* option is used to override the default extensions which are searched when trying to locate a file specified on a `EXEC SQL INCLUDE` line.

The default behavior when `EXEC SQL INCLUDE "FOO";` is encountered depends on the language of the source file. For C++, the following filenames will be tried in order for each directory in the –*I* search path:

```
FOO
FOO.sqC
FOO.sqx
FOO.hpp
FOO.h
```

For C, these filenames will be searched:

```
FOO
FOO.sqc
FOO.h
```

And for ASM, these filenames will be searched:

```
FOO
FOO.mac
```

However, if you specify –*macext ext* then `FOO.`*ext* will be searched instead. This way you can override the default extensions if your include/macro library uses different conventions.

The –flowerinc / –fnolowerinc options (enable/disable lowercasing of include filenames)

By default, **DB2PPC** will leave filenames as they are specified in the `EXEC SQL INCLUDE` command. However, for case-sensitive filesystems, it may sometimes be necessary to reduce all filenames to a uniform lower case, so that `EXEC SQL INCLUDE "FOO"` will search for `foo.h` instead of `FOO.h`. –*flowerinc* will cause this translation to occur.

The –fmrc / –fnomrc option (enable/disable use of mainframe-style return code)

The –*fmrc* option specifies that the translator should use mainframe-style return codes to indicate the exact error level reached. The –*fnomrc* option specifies that the translator should use Unix-style return codes that are either 0 (success) or 1 (errors). On OS/390 –*fmrc* is the default, while on cross-platform hosts –*fnomrc* is the default.

The –fnewfun / –fnonewfun option (enable/disable support for V8 NEW-FUN mode)

By default, **DB2PPC** supports the **NEWFUN** mode of DB2 V8 and produces DBRM output using UTF-8 character encoding and uses some extended structures. If *–fnonewfun* is specified, however, then output compatible with DB2 V7 is produced, using EBCDIC encoding for the DBRM output.

The –fattach=*mode* option (specify attachment mode)

The *–fattach=mode* option is used to specify the function for **DB2PPC** to use to attach to the DB2 server. There are three valid settings:

`tso`	Use `DSNHLI` (default).
`caf`	Use `DSNHLI2`.
`rrsaf`	Use `DSNHLIR`.

The –fconnect=1 / –fconnect=2 option (specify type for `CONNECT` statement)

If *–fconnect=1* is specified then **DB2PPC** generates the type 1 form of the `CONNECT` statement. By default, the type 2 form is generated.

The –fdate=*format* (specify a date format to override the system default)

The *–fdate=format* option specifies a format to use for date query results that will override the DB2 default. The following values are supported: *–fdate=iso*, *–fdate=usa*, *–fdate=eur*, *–fdate=jis*, *–fdate=local*. The default is for no date override to be in place, which can be manually specified with *–fdate=default*.

The –ffloat=ieee / –ffloat=s390 option (specify type of floating point variables)

If *–ffloat=ieee* is specified then **DB2PPC** notes that floating point host variables are in IEEE format instead of IBM's HFP format. The default is *–ffloat=s390*.

The –fpadntstr / –fnopadntstr option (enable/disable padding of NUL-terminated strings)

The *–fpadntstr* causes **DB2PPC** to note that NUL-terminated strings should be padded out to their full field length. The default is *–fnopadntstr*, in which case a NUL is used to indicate the end of data in the string regardless of length.

The –fsql=all / –fsql=db2 option (specify which SQL dialect to accept)

The *–fsql=all* option tells **DB2PPC** to accept all valid SQL syntax, even if it is not necessarily valid DB2 syntax. The default is *–fsql=db2*.

The –fstdsql option (use SQL rules for statements)

The *–fstdsql* option changes the `SQLCA` area to allow SQL rules for statements.

The –fversion=*ver* option (set the version field in the DBRM header)

The *–fversion=ver* option causes the specified string to be placed in the DBRM extended version header. The string is also placed in the C variable `SQLVERS.VERSSTR`.

The –fcpp option (enable C++ mode)

The *–fcpp* option causes the preprocessed output to be compatible with C++ compilers. The default is to output code compatible with C compilers.

The –fonepass / –ftwopass options (select one- or two-pass preprocessor operation)

The *–fonepass* option is the default and causes **DB2PP** to process the source in a single linear pass, so all declarations must come before references. The *–ftwopass* option causes **DB2PP** to process the source file in two passes, first reading declarations, and the second pass generating output.

The –ffold / –fnofold options (enable/disable folding identifiers to uppercase)

When *–ffold* is specified, all non-string identifiers will be converted to uppercase. *–fnofold* is the default option for C source. *–ffold* is the default option for ASM source.

The –fprog=*name* option (override the DBRM "program name" header field)

The **DBRM** output file has a "program name" header field which must match the member name of the DBRM when it is bound. Otherwise the **DB2 BIND** step

will produce a "consistency error." By default, the program name is derived from the DBRM output filename. If this does not match the actual names used on MVS, you may use *–fprog=name* to override the default.

The –fuser=*name* option (override the DBRM "user name" header field)

The **DBRM** output file has a "user name" header field. By default, your user name is determined using the value of the USER environment variable, but you can override it with the *–fuser=name* option.

The –ftimestamp=*hex* option (override the DBRM "timestamp" header field)

The **DBRM** output file has a timestamp field that the DB2 server uses to verify that different pieces come from the same build as a consistency check. If your build environment gives you cause to mix and match these pieces then you will need to manually ensure that your timestamps match using this option. This option takes a hex argument that may be up to 8 bytes (16 hex nibbles) long, i.e. `-ftimestamp=123456789ABCDEF0`.

The –v option

The *–v* option causes **DB2PPC** to produce a banner indicating the version and copyright information, then to exit with a return code of 0.

The –fdb2ver=v*N* option (set DB2 compatibility version)

The *–fdb2ver=vN* option instructs **DB2PPC** to emulate compatibility with the specified version of DB2. The supported values for *N* are 7, 8, 9, and 10. *–fdb2ver=v8*, and that is where the support is most complete.

The –fccsid=*num* option (set CCSID)

Many translated `EXEC SQL` commands specify the CCSID to use for database operations. That value can come from an `EXEC SQL DECLARE VARIABLE` statement, or it can come from the default value provided with this commandline option. If *–fdb2ver=v9* or *–fdb2ver=v10* is specified, then *–fccsid=500* is implied if no explicit *–fccsid=num* is present. Otherwise, for compatibility with older versions of DB2, the default is *–fccsid=37*.

The CCSID provided by *–fccsid=num* does not affect the source code's character set.

D2S

Using D2S

D2S is used to convert assembly `DSECT`s into C `struct` definitions. **D2S** examines `ADATA` information generated by the Systems/ASM assembler, **DASM**, or by IBM's `HLASM` assemble. **D2S** extracts `DSECT` information from the ADATA file generating a C structure definition suitable for use in a C or C++ header (".h") file.

For example, if the `EXAMPLE DSECT` was defined as in the following assembly source:

```
EXAMPLE DSECT
NEXT    DS  A       Address of next element
NAME    DS  CL30    Customer name
ID      DS  F       Customer ID number
        END
```

then **D2S** would convert that `DSECT` into the following C structure:

```
struct example {
  void * __ptr31 next;                  /* Address of next element */
  unsigned char  name[30];              /* Customer name */
  char           __filler0[2];
  int            id;                    /* Customer ID number */
};
```

Notice that **D2S** preserved the original comments, and has inserted filler in order to ensure the fields match exactly with the original `DSECT` definition.

For more information on how to use **DASM** to generate the `ADATA` information required by **D2S**, refer to the *Systems/ASM Assembler* manual. For more information on how to use IBM's `HLASM`, refer to IBM *HLASM V1R4 Programmer's Guide* document number SC26-4941-03.

A simple UNIX or Windows **DASM** command to generate `ADATA` information in the file `adata` would be

```
dasm -A=adata example.asm
```

On Windows and Unix, **D2S** is executed with a command line of the form

```
d2s [options] input file
```

When run on OS/390 or z/OS, options may be specified in the `PARM` statement. If no input file is specified, the `SYSIN` DD is used. Output defaults to the `EDCDSECT` DD. Informational and error messages are output to `STDOUT` and `STDERR` DDs. The following JCL could be used on OS/390:

```
//D2S       JOB
//D2S       EXEC PGM=D2S,PARM='options'
//STEPLIB   DD DSN=Systems/C load library,DISP=SHR
//STDERR    DD SYSOUT=*
//STDOUT    DD SYSOUT=*
//EDCDSECT  DD SYSOUT=*
//SYSIN     DD DSN=ADATA(FOO),DISP=SHR
```

D2S Options

–help	display usage information
–s *sect_name*	specify section (DSECT) to convert
–com=*opt* / –xcom	control inclusion of comments
–anon / –xanon	enable/disable the use of anonymous substructures/unions
–def / –xdef	enable/disable `#define` directives for substructures/unions
–equ=*opt* / –xequ	control how EQU symbols are converted
–style=*kind*	control the style of the generated C structures
–in=*count* / –xin	control indentation
–lc / –xlc	enable/disable converting names to lower case
–elc / –xelc	enable/disable converting EQU names to lower case
–ll=*count* / –xll	set a maximum output line length
–pp / –xpp	include preprocessor directives to prevent duplicate definitions
–seq / –xseq	enable/disable sequence numbers on the output
–unique=*str* / –xunique	specify a unique string for building labels
–out=*filename*	specify a different output filename
–char=*type*	specify the type to be generated for `DS C` members
–prefixmap=*old,new*	specify a string mapping to be performed on member names
–lengthsuffix=*suf*	specify unique suffix for length #defines from –equ=def
–v	print version information
–ds0xl	accept `DS 0XL(n)` as a synonym for `EQU n`

The –help option (display usage information)

This option causes **D2S** to output basic usage information on the `stdout` stream.

The –s *sect_name* option (specify section to convert)

The *–s* parameter specifies that **D2S** should convert the DSECT named *sect_name*. If the *–s* parameter is not specified then all DSECTs will be converted.

The –com=*opt* and –xcom options (control inclusion of comments)

Control which comments will be included in the generated C source code. There are three possible values for *opt*:

    `single` Include comments that appear on the `DS` line.

    `cont`   Include those comments plus continuations of that `DS` line.

    `all`   Include all comments, including ones that begin their own line.

If *–xcom* is specified then no comments will be included at all. The default is *–com=single*.

The –anon and –xanon options (control the use of anonymous substructures/unions)

Systems/C and Systems/C++ offer an *–fanonstruct* option that provides the capability for `union` and `struct` members of another `struct` to be unnamed (and the members of the anonymous structures and unions members are then considered to be members of the enclosing structure). *–anon* is the default and causes **D2S** to generate C code which uses anonymous substructures and unions. Consider the following DSECT example:

```
TEST DSECT
A     DS 0F
A1    DS CL1
A2    DS CL1
A3    DS CL1
A4    DS CL1
```

With *–xanon* it converts to the following C structure:

```
struct test {
  union {
    int             a;
    struct {
      char             a1;
      char             a2;
      char             a3;
      char             a4;
    } __struct0;
  } __union0;
};
```

Whereas *–anon* causes the following C structure to be generated:

```
struct test {
  union {
    int             a;
    struct {
      char             a1;
      char             a2;
      char             a3;
      char             a4;
    };
  };
};
```

The –def and –xdef options (control #define directives)

The *–def* and *–xdef* options control the generation of `#define` directives for sub-structures/unions. *–def* is only useful if *–xanon* is also specified, in which case the following lines would be generated for the *–xanon* example:

```
#define a __union0.a
#define a1 __union0.__struct0.a1
#define a2 __union0.__struct0.a2
#define a3 __union0.__struct0.a3
#define a4 __union0.__struct0.a4
```

–xdef is the default.

The –equ=*opt* and –xequ options (control EQU conversion)

The *–xequ* option is the default and instructs **D2S** to discard EQU symbols. There are four possible values for *opt*:

> def   Output C `#define`s for EQU symbols.
>
> sym   Output C symbols for EQU symbols.
>
> bit   Output C bitfields for EQU symbols, using the value of the EQU as a bit mask.
>
> bitl   Like `bit`, except using the length as a bit mask.
>
> mixed   Like `bit`, except symbols which do not work as bitfield masks are output as `#define`s rather than discarded.

For example, consider the DSECT:

```
TEST DSECT
BF   DS CL2
FLG  EQU 4,5
FLGS EQU 3
```

If *–equ=def* is specified then the following C structure is generated:

```
struct test {
  unsigned char  bf[2];
  #define flg 4
  #define flg_length 5
  #define flgs 3
};
```

Note that the length attribute which was provided for `FLG` appears in the C code as a separate `#define` with "_length" appended to its name. The suffix can be specified with *–suffixlength=suf*, which may be helpful if you are experiencing namespace collisions due to the length symbols.

If you use *–equ=sym* then **D2S** will treat the EQU values as offsets into the structure, generating:

```
struct test {
  unsigned char  bf[2];
  char           __filler0[1];
  unsigned char  flgs;
  unsigned char  flg[5];
};
```

If –*equ=bit* is specified then the following C structure is generated instead:

```
struct test {
  /* bitfield for bf: */
  unsigned int   __filler0 : 13;
  unsigned int  flg : 1;
  unsigned int  flgs : 2;

};
```

For –*equ=bitl* consider the following DSECT:

```
TEST DSECT
A    DS CL4
B    DS CL3
BF   DS CL2
FLG  EQU A
FLGS EQU B
```

which is converted to:

```
struct test {
  unsigned char  a[4];
  unsigned char  b[3];
  /* bitfield for bf: */
  unsigned int   __filler0 : 13;
  unsigned int  flg : 1;
  unsigned int  flgs : 2;

};
```

`FLG` and `FLGS` both inherit length attributes from `A` and `B`, and those lengths are interpretted as if they were bitmask values specifying bitfields within `BF`.

The –style=*kind* options (control the style of the generated C structures)

The –*style=* option provides control over the style of C structures **D2S** produces.

The valid styles are *fold*, *nest*, *nestorg*, and *raw*.

When the *raw* style is specified, **D2S** does not attempt to reorganize the generated structures in any fashion. Although this will result in a faithful reproduction of the

assembly language DSECT, the C source can be unpleasant to read if the DSECT is at all complicated. There will be many `union` and `struct` fields with appropriate filler bytes defined to ensure everything is placed at the correct offset. Also, when *–style=raw* is specified, the structure will be "flat" with no nested sub-structures.

When the *fold* style is specified, **D2S** will reorganize structures to try to remove unnecessary filler fields. This can make the structure quite different in order from the orginal assembly language source.

When the *nest* style is specified, **D2S** will again attempt to reorganize the defined structure to remove unnecessary filler fields. However, it will attempt to recognize common assembly-language constructs that indicate one element of the DSECT is a "sub-structure" of the others. In doing so, it will retain the same order of definition in the generated C structure as was discovered in the original assembly language source. The result of the conversion when *–style=nest* is specified are typically more "C like" while retaining the intent of the original source.

When the *nestorg* style is specified, **D2S** will use the same technique for re-ordering the members as if *nest* were specified, except that it will use a slightly different heuristic for `ORG` mnemonics. The result is often much more sensible.

For example, given the following DSECT:

```
TEST DSECT
A    DS F     OFFSET 0
B    DS F     OFFSET 4
     ORG B
C    DS F     OFFSET 4
D    DS F     OFFSET 8
     ORG D
E    DS F     OFFSET 8
     END
```

if *–style=raw* is specified then the following C structure will be generated:

```
struct test {
  union {
    struct {
      int          a;                        /* OFFSET 0 */
      int          b;                        /* OFFSET 4 */
    };
    struct {
      char         __filler0[4];
      int          c;                        /* OFFSET 4 */
      int          d;                        /* OFFSET 8 */
    };
    struct {
      char         __filler1[8];
      int          e;                        /* OFFSET 8 */
    };
  };
};
```

when *–style=fold* is specified, the following C structure will be generated:

```
struct test {
  int            a;                          /* OFFSET 0 */
  union {
    struct {
      int          b;                        /* OFFSET 4 */
      int          e;                        /* OFFSET 8 */
    };
    struct {
      int          c;                        /* OFFSET 4 */
      int          d;                        /* OFFSET 8 */
    };
  };
};
```

when *–style=nest* is specified, the following, more nested C structure will be generated:

```
struct test {
  int              a;                              /* OFFSET 0 */
  union {
    int                b;                            /* OFFSET 4 */
    struct {
      int                  c;                          /* OFFSET 4 */
      union {
        int                    d;                        /* OFFSET 8 */
        int                    e;                        /* OFFSET 8 */
      };
    };
  };
};
```

when *–style=nestorg* is specified, it will be a little more tempered in its nesting:

```
struct test {
  int              /*  FL4 */ a;            /* OFFSET 0 */
  union {
    int                /*  FL4 */ b;          /* OFFSET 4 */
    int                /*  FL4 */ c;          /* OFFSET 4 */
  };
  union {
    int                /*  FL4 */ d;          /* OFFSET 8 */
    int                /*  FL4 */ e;          /* OFFSET 8 */
  };
};
```

The –in=*count* and –xin options (control indentation)

If *–in=count* is specified then *count* spaces will be used in nested members of C structures. *–xin* is equivalent to `-in=0`. The default is `-in=2`.

The –lc and –xlc options (control case conversion)

If *–lc* (the default) is specified then all symbol names (except for EQU names) are converted to lower case by **D2S**. *–xlc* specifies that the names are to remain in their original form.

The –elc and –xelc options (control EQU case conversion)

If *–elc* (the default) is specified then the names of EQU symbols are converted to lower case by **D2S**. *–xelc* specifies that the EQU names are to remain in their original form. *–xelc* may be desirable if *–equ=def* is used, as C programmers often prefer that `#define` constants have uppercase names.

The –ll=*count* and –xll options (control output line length)

The *–ll=count* option specifies that the output lines are to be no wider than *count* columns. *–xll* removes this limit and is the default.

The –pp and –xpp options (control guard preprocessor directives)

When *–pp* is specified, **D2S** will generate preprocessor directives to guard against duplicate definitions of the same structure. *–xpp* is the default and specifies that no such directives are to be generated.

The directives for an example structure would look like:

```
#ifndef example__
#define example__

struct example {
   ...
};

#endif  /* example__ */
```

The –seq and –xseq options (control sequence number output)

The *–seq* option causes **D2S** to output sequence numbers in columns 73-80, implying `-ll=72`. The default is *–xseq* which specifies that no sequence numbers are to be output.

The –unique=*str* and –xunique options (control label building)

When **D2S** encounters a symbol with characters which C does not accept, it replaces them according to the following table:

Character	Replacement value
@	*str* a *str*
#	*str* n *str*
$	*str* d *str*

The default for *str* is the empty string, which can be specified with *–xunique*. If `-unique=_` were specified then the label `X$Y` would be converted to the C symbol `X_d_Y`.

The –out=*filename* option (specify output filename)

The *–out=filename* option specifies that the C output should be directed to *filename*. If no *–out=filename* is specified then on Windows or Unix ".h" is appended to the name of the input file and used as the output file. On MVS `//DDN:EDCDSECT` is the default.

The –char=*type* option (specify the type to be generated for `DS C` members)

By default when a DSECT member defined as `DS C` is encountered, a C member with type `unsigned char` is generated. However, sometimes it may be necessary to use a different character type for compatibility with other declarations, especially in C++. If *–char=char* is specified then members of type `char` are generated. If *–char=signed* is specified then members of type `signed char` are generated. If *–char=unsigned* is specified then members of type `unsigned char` are generated.

For some cases such as oddly-sized members of a non-character type (i.e., `DS FL5`), an `unsigned char` member will be generated regardless of the setting of the *–char=type* option. These members are essentially placeholders and do not properly represent the meaning of the underlying type.

The –prefixmap=*old,new* option (specify a string mapping to be performed on member names)

Since `DSECT` members are all in the global namespace, many assembly programmers use a common prefix for all members of the same `DSECT`. The *–prefixmap=old,new* option allows you to replace the prefix string with a new string. If the *new* string is not provided, i.e., *–prefixmap=str*, then any prefix of *str* on a member name is stripped. The *–prefixmap* option may be specified more than once, to specify several different prefixes. The prefix mapping is applied before any other translation on the member names (such as lowercasing, or @, #, $ translation), so your *old* string must exactly match the prefix used in the assembly source.

The –lengthsuffix=*suf* option (specify unique suffix for length #defines from –equ=def)

When *–equ=def* is specified, an extra `#define` is generated for EQUs that have an assigned length attribute. This `#define` is given a name which consists of the EQU name followed by the length suffix, which defaults to `_length`. Since this could potentially cause namespace conflicts with existing symbol names, the *–lengthsuffix=suf* option allows you to specify a different suffix which will be used instead.

The –v option

The *–v* option causes **D2S** to produce a banner indicating the version and copyright information, then to exit with a return code of 0.

The –ds0xl option (accept `DS 0XL(n)` as a synonym for `EQU n`)

When *–ds0xl* is specified, any `DS 0XL(n)` statements in the ASM source are interpretted as if they were `EQU n` instead (or `EQU 0,n` if *–equ=bitl* is in effect).

ASCII/EBCDIC Translation Table

The Dignus utilities use the following tables to translate characters between ASCII and EBCDIC. These tables represent the mapping of the IBM Code Page 1047 to ISO LATIN-1.

ASCII to EBCDIC

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	37	2D	2E	2F	16	05	15	0B	0C	0D	0E	0F
1	10	11	12	13	3C	3D	32	26	18	19	3F	27	1C	1D	1E	1F
2	40	5A	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
3	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
4	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
5	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	AD	E0	BD	5F	6D
6	79	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
7	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	C0	4F	D0	A1	07
8	20	21	22	23	24	25	06	17	28	29	2A	2B	2C	09	0A	1B
9	30	31	1A	33	34	35	36	08	38	39	3A	3B	04	14	3E	FF
A	41	AA	4A	B1	9F	B2	6A	B5	BB	B4	9A	8A	B0	CA	AF	BC
B	90	8F	EA	FA	BE	A0	B6	B3	9D	DA	9B	8B	B7	B8	B9	AB
C	64	65	62	66	63	67	9E	68	74	71	72	73	78	75	76	77
D	AC	69	ED	EE	EB	EF	EC	BF	80	FD	FE	FB	FC	BA	AE	59
E	44	45	42	46	43	47	9C	48	54	51	52	53	58	55	56	57
F	8C	49	CD	CE	CB	CF	CC	E1	70	DD	DE	DB	DC	8D	8E	DF

EBCDIC to ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	02	03	9C	09	86	7F	97	8D	8E	0B	0C	0D	0E	0F
1	10	11	12	13	9D	0A	08	87	18	19	92	8F	1C	1D	1E	1F
2	80	81	82	83	84	85	17	1B	88	89	8A	8B	8C	05	06	07
3	90	91	16	93	94	95	96	04	98	99	9A	9B	14	15	9E	1A
4	20	A0	E2	E4	E0	E1	E3	E5	E7	F1	A2	2E	3C	28	2B	7C
5	26	E9	EA	EB	E8	ED	EE	EF	EC	DF	21	24	2A	29	3B	5E
6	2D	2F	C2	C4	C0	C1	C3	C5	C7	D1	A6	2C	25	5F	3E	3F
7	F8	C9	CA	CB	C8	CD	CE	CF	CC	60	3A	23	40	27	3D	22
8	D8	61	62	63	64	65	66	67	68	69	AB	BB	F0	FD	FE	B1
9	B0	6A	6B	6C	6D	6E	6F	70	71	72	AA	BA	E6	B8	C6	A4
A	B5	7E	73	74	75	76	77	78	79	7A	A1	BF	D0	5B	DE	AE
B	AC	A3	A5	B7	A9	A7	B6	BC	BD	BE	DD	A8	AF	5D	B4	D7
C	7B	41	42	43	44	45	46	47	48	49	AD	F4	F6	F2	F3	F5
D	7D	4A	4B	4C	4D	4E	4F	50	51	52	B9	FB	FC	F9	FA	FF
E	5C	F7	53	54	55	56	57	58	59	5A	B2	D4	D6	D2	D3	D5
F	30	31	32	33	34	35	36	37	38	39	B3	DB	DC	D9	DA	9F