Technical Addendum for HP Link Editor/iX

HEWLETT PACKARD

HP Part No. 32650-90845
Printed in U.S.A. October 1995

1
E1095

Contents

1. **Technical Addendum for HP Link Editor/iX**

A Technical Addendum Is: 1
Overview 1
Dependent Libraries 2
  Example 2
New Commands for Maintaining Executable Libraries 3
  ALTXL Command 3
    Syntax: 3
    Parameters 3
    Examples 4
  BUILDXL Command 5
    Syntax 5
    Example: 5
Shared Global Data 6
  LKSHAREDATA Variable 7
Compiling for Shared Global Data 8
  Global Data Limit 8
Linking for Shared Global Data 10
  SHARE Option 10
  New LIBC/iX Library 11
    Linking a C Application with C Libraries 11
    Guidelines for the Linkage Method 12
  POSIX Links 12
  Hiding Data Symbols 12
  Linking With Hidden Data Symbols 13
  Revealing Data Symbols 14
Building a Shared Global Data XL 15
  Adding an Object File to an XL 15
  Manipulating XLs 17
  Listing Program Files and XLs 18

Loading a Shared Global Data Program File or XL 20
Loader Binding Rules 20
Compatibility 21
Linking with Compatibility Mode Files 21
Advanced Topics 22
External Data References 22
Duplicate Data 23
Sample Listings 26
Symbol Transformation Example 26
LISTPROG Example 28
LISTXL Example 30
Diagnostic Messages 32
User Errors (1000-1499) 32
Warning Messages (1500-1999) 32
Language Subsystem Errors (3000-3999) 33

Glossary

Tables

1-1. Shared Global Commands or Syntax 6
1-2. Library Symbol Table 7
1-3. Compiler Options for Additional External Data References . . 9

Technical Addendum for HP Link Editor/iX

A Technical Addendum Is:

A Hewlett-Packard technical addendum is an informal document that delivers the latest information to customers and support personnel before formal revisions to the HP manual sets. The characteristics of these addendums are listed below:

- Technical addendums are published when significant improvements are made to an HP product.
- Technical addendums supplement the information contained in the existing manual set. They are not sets of replacement pages.
- Information from the technical addendum will be incorporated into the revision of the manual set that is printed after the issue date of the addendum.
- Keep these pages with your existing HP documentation until the next formal revision of the manual set.

Overview

This document describes changes made to the HP Link Editor for the HP 3000 Series 900 MPE/iX 5.5 release for dependent libraries and sharing global data in executable libraries (XLs) and programs (NMPRGs). It complements the information contained in the *HP Link Editor/iX Reference Manual*.

Dependent Libraries

Dependent libraries are a new feature available on MPE/iX. They are libraries that must be loaded in addition to the executable library (XL) currently loaded.

When a program runs and the loader loads each XL, it checks the dependent library list for that XL and loads any dependent libraries that have not been loaded already. This allows the dependencies to be isolated within the libraries. You only need to specify the first-level libraries when linking or running the program.

Dependent libraries can be specified when you build the XL or by using the new ALTXL command (*see below*) to alter an existing XL's dependent library.

Example

If an XL named `lib1` was built with dependent libraries `lib2` and `lib3`, and `lib2` had its own dependent libraries `lib4` and `lib5`, the load graph would look like the following:

```
lib1 -> lib2 -> lib4 -> lib5 -> lib3 -> XL.PUB.SYS
```

New Commands for Maintaining Executable Libraries

ALTXL Command

This `ALTXL` command changes the dependent library string for an executable library (XL). It accepts two options, both are required: the target XL name and the dependent library string. Dependent libraries can be specified in an indirect file or directly in the argument to the `LIB=` parameter.

When the Link Editor builds an XL, it inserts an MPE Program Auxiliary Header after the LST header. Any dependent library string for the XL is inserted in this header.

The dependent library string algorithm for `ALTXL` is similar to the `ALTPROG` string algorithm. If a new dependent library string is the same size or smaller than the current dependent library string in the XL, the Link Editor overwrites the old string. If the new string is larger than the old string, the Link Editor attempts to write the new string at the end of the LST string table. If there is not enough room in the string table, an error message is displayed. If this happens, you can only add or change dependent library strings in the XL by rebuilding the XL.

Syntax:

`ALTXL XL=`*xl_name*`; LIB=`*dependent_library*

Parameters

xl_name	Names the executable library whose dependent library list is to be altered.
dependent_library	Names a list of dependent libraries that must be loaded when this XL is loaded. Each dependent library must have a filecode of `NMXL`. When you want to include several libraries, you can name each library directly, or you can name a single file that contains a list of the libraries you want to include. If you use this last, indirect method, you must precede the indirect file name with a caret symbol (^).

Note

You must supply at least one dependent library since the `LIB=` parameter is required.

Examples

```
:PRINT MYINDF1
       NEWLIB.LIB.MYACCT
       LIB2.LIB.SYS

:LINKEDIT
LinkEd> ALTXL MYLIB1; LIB=^MYINDF1
LinkEd> ALTXL NEWLIB.LIB.MYACCT; LIB=NEW2.TMP.SYS
LinkEd> EXIT

:RUN MYOUT;XL="MYLIB1"
```

The loader loads the program file, `MYOUT`, then the executable library, `MYLIB1`. Since `MYLIB1` has dependent libraries, `NEWLIB.LIB.MYACCT` and `LIB2.LIB.SYS`, the loader loads them. `NEWLIB.LIB.MYACCT` has a dependent library `NEW2.TMP.SYS` so it is loaded after `NEWLIB.LIB.MYACCT` and before `LIB2.LIB.SYS`. The load graph is as follows:

```
MYOUT->MYLIB1->NEWLIB.LIB.MYACCT->NEW2.TMP.SYS->LIB2.LIB.SYS
```

```
:LINKEDIT
LinkEd> ALTXL MYLIB2;LIB=MYXL.PUB.LINKER, MYXL2.TMP.SYS
LinkEd> EXIT

:RUN MYOUT;XL="MYLIB2"
```

When `MYLIB2` is loaded, the loader loads `MYXL` and `MYXL2`. If either of these libraries has its own dependent libraries, each new dependent library is loaded, in order, after the library that contains the dependency.

BUILDXL Command

This command now accepts a third parameter, `LIB=`, used to specify dependent libraries. This parameter accepts an indirect file or a list of fully qualified library names. The Link Editor concatenates the dependent library names into a string and inserts the string into the LST Program Auxiliary header for the loader to search at run time.

Syntax

`BUILDXL XL=`*xl_file*

[`;LIMIT=`*max_modules*]

[`;LIB=`*dependent_library*]

Where:

dependent_library	Names a dependent library or a list of dependent libraries that must be loaded when this XL is loaded. Each dependent library must have a filecode of `NMXL`. When you want to include several libraries, you can name each library directly, or you can name a single file name that contains a list of the libraries you want to include. If you use this last, indirect method, you must precede the indirect file name with a caret symbol (^).

If the `LIB=` option is not specified, the XL will be built without any dependent libraries.

Example:

```
BUILDXL MYXL2;LIB=^MYINDF
```

This command creates an executable library named `MYXL2`, that will have dependent libraries as specified in `MYINDF`. When `MYXL2` is loaded, its dependent libraries are also loaded.

Shared Global Data

Shared global data are data definitions, initialized or uninitialized, that are exported to or imported from other compilation units. The data definitions are visible to other modules if the definitions are exportable. This means that they are not marked as *hidden* and, therefore, not ignored by the loader. Table 1-1 shows the languages and the appropriate shared global command and syntax.

Table 1-1. Shared Global Commands or Syntax

Language	Command

HP Pascal/iX	`$GLOBAL$`
	`$EXTERNAL$`
	Module variables

HP C/iX	globals
HP FORTRAN/iX	common
HP COBOL II/iX	`EXTERNAL` clause
HP RPG/iX	`RLABL`

Note Shared global data as defined in this addendum means each process will have its own copy of the data. This is *not* to be confused with all processes sharing one copy of the data.

Each program file and executable library (XL) contains a Library Symbol Table (LST) at the beginning of the file. Normally, during linking, the Link Editor places unresolved references to code in an import list. With shared global data, the Link Editor also places unresolved references to data and data exports in the import list. At run time, the loader resolves the symbols in the import list by searching the LST in the program file and in one or more XLs. The loader builds an External Reference Table (XRT) that tracks externally-called procedures and allows them to be shared.

Additionally, the loader initializes a Data Cross Reference Table (DXRT) with data definition addresses. The DXRT holds entries for all data references that need a corresponding definition. This allows externally referenced data to be tracked and shared.

Table 1-2 shows the data types and scopes that are inserted into the Library Symbol Table.

Table 1-2. Library Symbol Table

Data Types	**Scope**
Data Universal	An initialized data symbol defined in an object module that is visible (exportable) to other object modules.
Data Unsat	A data symbol that is referenced by an object module but not defined in it. For shared global data, these symbols are now allowed in XLs and program files.
Storage Universal	An uninitialized data symbol created when an uninitialized data declaration is made. Note that if no corresponding Data Universal is found for it during linking, the scope is changed from Unsat to Universal and the symbol is exportable and can now be used to satisfy references (imports).

LKSHAREDATA Variable

You can now set a *job* or *session* variable called `LKSHAREDATA`. If `LKSHAREDATA` is set to `TRUE`, all subsequent linking is for shared code and data. This is equivalent to linking with the `;SHARE` option.

If `LKSHAREDATA` is not set or is set to `FALSE`, all subsequent linking is for *shared code only*. If you want to share data, the `;SHARE` option must be specified on the link line.

Caution

Setting this variable affects *all* linking done in the job or session. There is no option to turn off sharing data during a link if this variable is set to `TRUE`. If `logon` UDCs are used, this variable can be set for each user, all users in an account, or all users on the system.

Compiling for Shared Global Data

To share global data across object modules, you do not need to recompile an application. You only need to relink the application. Refer to “Linking for Shared Global Data”.

Global Data Limit

There is a limit to the number of data references (approximately 2000) allowed per program file and per module in an XL. If this limit is exceeded, the Link Editor emits an error message instructing you to recompile the appropriate modules to allow more external data references.

Refer to “Diagnostic Messages” for the error message.

Use the `LISTPROG` command with the `;DATA` and `;VALUE` options to list the data symbols in a program file. Count the number of Data Unsats listed to determine how close the program file is to the limit.

Use the `LISTXL` command with the `;DATA` and `;VALUE` options to list the data symbols in an XL. Count the number of Data Unsats listed *per module* to determine how close a module is to the limit. If an XL is close to or over the limit, you can restructure the module so that it does not go over the limit for data references. You can also hide symbols with `HIDERL`; refer to “Hiding Data Symbols” for more information. If you do not restructure your modules, you must recompile them using one of the new compiler options.

Table 1-3 lists the new compiler options available to allow more external data references.

Table 1-3.
Compiler Options for Additional External Data References

Option	Languages
`$MORE_GLOBALS ON$`	MPE/iX compiler option available on HP Pascal/iX.
`$CONTROL MOREGLOBALS`	MPE/iX compiler option available on HP COBOL II/iX.
`+k`	Command-line option, available on HP C/iX.

Note

If an application needs to be recompiled and the compiler does not support one of the new compiler options, the application can either be restructured to have fewer global data references or relinked without specifying the `SHARE` option. Refer to "SHARE Option" for more information.

Linking for Shared Global Data

To link a program file for shared global data, it is not necessary to specify RL files at link time if you want any unresolved data references to be satisfied at load time by an XL. The default behavior remains linking for non-shared data. To link for shared data, you must use the new `SHARE` option. The loader automatically searches XL.PUB.SYS at run-time if you did not specify it at link time. You can override an XL specified at link time by specifying the XL at run time using the `XL=` option in the `RUN` command.

SHARE Option

The `SHARE` option instructs the Link Editor to link for shared data. Linking for shared data means all non-hidden Data Universals and Storage Universals are exported, and all Data Unsats and Storage Universals are imported. If you want to selectively export Data Universals and Storage Universals, you can take the definitions out of the source code of the program file or library and add the objects to an RL. Use the `HIDERL` command to hide the definitions and add the resulting RL to an XL. At link time, specify the XL on the link line so the loader will know it needs to search that XL first. The data symbols you want hidden are not exportable from the XL or program file.

Example

```
 LINK FROM=^indirect_file;SHARE;XL=MYXL;TO=MYOUT;RL=LIBCSHR.LIB.SYS
```

The object files in `indirect_file` and the RL `LIBCSHR` are linked to form an executable program file called `MYOUT`. At run time, the loader loads the XL `MYXL` followed by `XL.PUB.SYS` and attempts to satisfy unresolved data references in `MYOUT` with data exports in `MYXL` and `XL.PUB.SYS`. The loader issues an error message if it cannot resolve a data reference.

Note

If you are using C and want to link for shared global data, you must link with `LIBCSHR` (see "New LIBC/iX Library," following). If you do not want to link for shared global data you can continue to link with `LIBCINIT`.

New LIBC/iX Library

A new C library, `libcshr.lib.sys`, takes advantage of the Shared Global feature. This library allows you to link your program with other executable libraries (XLs) in addition to `xl.pub.sys`. You can use it in the same way you use other `libc` libraries, but you must have the new share keyword. For example:

```
link from=myobj;to=myprog;rl=libcansi.lib.sys,libcshr.lib.sys;share
```

This links an ANSI-conforming application `myobj` into a program file *myprog* with all `libc` functions and global variables resolved in `xl.pub.sys`.

Note The keyword *share* must be specified since the default linking mode is not shared.

Linking shared programs results in a much smaller application and allows you to take advantage of any new system releases without having to relink. Existing programs and scripts that use `libc.lib.sys` and `libcinit.lib.sys` continue to work unchanged. However, `libcshr.lib.sys` is the recommended library to use. The scripts `ccxllk.pub.sys` and `ccxlgo.pub.sys` link with `libcshr`.

Linking a C Application with C Libraries

There are three suggested methods for linking a C application and libraries:

1. *Archive Linking* - Link with *LIBC* which is nonsharable. All `libc` functions are completely integrated into the application.

   ```
   link from=cobj;to=cprog;rl=libcansi.lib.sys,libcinit.lib.sys
   ```

2. *Semi-Shared Linking* - Link with *LIBCINIT* which is designed to reinitialize the *XL.PUB.SYS* global pointers and redirect them to the program's globals at process startup. Most of the `libc` functions are in *XL.PUB.SYS* and are shared partly by the application.

   ```
   link from=cobj;to=cprog;rl=libcansi.lib.sys,libc.lib.sys
   ```

3. *Shared Linking* - Link with *LIBCSHR*, which is completely sharable. All the `libc` functions are in *XL.PUB.SYS*. They are shared by the application's linked executable libraries. It is the recommended method for sharing global data.

```
 link from= cobj;to=cprog;rl=libcansi.lib.sys,libcshr.lib.sys;share
```

Guidelines for the Linkage Method

- Use *non-shared* linkage for self-contained and highly portable applications. This is the traditional linkage method.
- Use *semi-shared* linkage for applications that want partial sharing but maintain a high degree of compatibility. This is the current shared linkage method.
- Use *shared* linkage for applications that want maximum sharing without any of the semi-shared linkage restrictions or workarounds. This is the new, true shared linkage method.

POSIX Links

All POSIX links using `c89` or `/lib/libc.a/` are linked for shared code and data.

Hiding Data Symbols

You are now able to hide data symbols as well as code symbols in RLs. The `HIDERL` command sets a bit called the *hidden bit* for each symbol marked as hidden in an RL. When the RL is added to an XL or program file, the Link Editor does not build LST export records for these symbols. The symbols are not visible or accessible to modules in the same XL, other XLs, or program files.

There are two options to the `HIDERL` command to support hiding data symbols:

`ALL_DATA`	Specifies that all Data Universals and Storage Requests are marked as hidden in the RL. The hidden bit is set in the symbol dictionary for these symbols.

`DATA=`*data_item*	Specifies that *data_item* is marked as hidden in the RL. Note that *data_item* is case-sensitive and must be a Data Universal or Storage Request.

As with code symbols, the hiding takes place *when the RL is added to the XL or program*. Therefore, if you link with an RL that has hidden symbols, those hidden symbols are still exportable to the program file and can satisfy imports in the program file. When the RL is added to an XL or program file, no LST import or export record is generated in the XL's LST area for the hidden symbols. The hidden symbols are therefore not sharable with other modules. Like Code Unsats, Data Unsats cannot be hidden even though the hidden bit is set.

Duplicate data symbols are allowed in RLs. When hiding data or storage symbols, the Link Editor hides each *occurrence* of the symbol. For example, if there are five data or storage symbols called `foo`, all five are marked as hidden.

Using `HIDERL` to hide data symbols is the recommended way to keep data symbols internal (that is, not externally visible) to the load module.

Example

```
HIDERL RL=MYRL;DATA=foo
HIDERL RL=MYRL2;ALL_DATA

ADDXL FROM=MYRL;TO=MYXL
ADDXL FROM=MYRL2;TO=MYXLA2
```

`MYXL` does not export `foo` because it is marked as hidden. `foo` is not visible to any other module in `MYXL`, other XLs, or program files. `MYXLA2` does not export any data symbols because all data symbols are marked as hidden.

Linking With Hidden Data Symbols

Because data in RLs can be hidden, when linking or doing an `ADDXL;MERGE` with modules that have hidden data symbols, if a hidden data symbol is processed, all occurrences of that symbol must also be marked as hidden. If one of the occurrences is not marked as hidden, the Link Editor emits a warning message and marks the symbol as hidden.

Refer to "Diagnostic Messages" for the warning message.

Revealing Data Symbols

You are now able to reveal, or unhide, data symbols as well as code symbols in RLs. The `REVEALRL` command unsets the hidden bit for each symbol specified. When the RL is added to an XL, the Link Editor exports all unhidden symbols. The symbols become visible and accessible to the same XL, other XLs, and program files.

There are two new options to the `REVEALRL` command to support unhiding data symbols:

`ALL_DATA`	Specifies that all Data Universals and Storage Requests are marked as unhidden in the RL. The hidden bit, if set, is unset for these symbols.
`DATA=`*data_item*	Specifies that *data_item* is marked as unhidden in the RL. Note that *data_item* is case-sensitive and must be a Data Universal or Storage Request.

As with code symbols, the revealing takes place *when the RL is added to the XL or program file*. When this happens, the symbols become sharable with other modules.

Duplicate data symbols are allowed in RLs. When revealing data or storage symbols, the Link Editor reveals each *occurrence* of the symbol. For example, if there are five data or storage symbols called `foo`, all five are revealed.

Example

```
REVEALRL RL=MYRL;DATA=foo
REVEALRL RL=MYRL2;ALL_DATA

ADDXL FROM=MYRL;TO=MYXL
ADDXL FROM=MYRL2;TO=MYXL2
```

`MYXL` exports `foo` because it is revealed, or unhidden. Modules in `MYXL`, as well as in other XLs or program files, can access `foo`. `MYXL2` exports all data symbols. Modules in `MYXL2`, as well as other XLs or program files, can access those data symbols.

Building a Shared Global Data XL

An XL can have shared global data modules and non-shared global data modules. You can determine if a module in an XL contains shared global data by using the `LISTXL` command. If the `SHARED DATA` header field is set to `YES`, the module is a shared global data module. If the `SHARED DATA` header field is set to `NO`, the module does not contain shared global data.

As before, the `BUILDXL` command builds an empty XL. The `ADDXL` command is used to add shared global data modules to an XL. RLs or object files can still be specified as input to the `ADDXL` command.

Note

XLs can have duplicate data exports.

Duplicate Storage Universals in an XL is acceptable. However, duplicate Storage Universals are ignored by all XL commands except the `LISTXL` command.

It is *not* recommended that XLs contain duplicate Data Universal exports. Most XL commands process Data Universal exports and, like entry points, it assumes Data Universal exports in the XL are unique.

XLs with duplicate module names containing duplicate Data Universal exports are *not* acceptable because there is no unique way for the XL commands to correctly identify which module to process.

Adding an Object File to an XL

To add object files to an XL and have it share global data, the `SHARE` option must be specified or the `LKSHAREDATA` variable must be set to `TRUE`. The default behavior remains linking for non-shared data. When adding modules to an XL and sharing is turned on, all non-hidden Data Universals and Storage Universals are exported and all Data Unsats, Data Universals, and Storage Universals are imported. The Universals are exported and imported to give the loader the chance to decide which export will be used as the single overriding definition that will resolve all outstanding references to the export for the life of the load. This is slightly different than when linking an object file to form a program file and sharing data. In that type of link, Data Universals are *not* importable because Data Universals in program files always take precedence over Data Universal exports in an XL. For more information on the loader binding rules, refer to "Loader Binding Rules".

The Link Editor now allows modules in an XL to contain unresolved data symbols when linking for shared code and data. At load time, the loader attempts to resolve these symbols. If the symbols are not resolved, the loader emits an error message and aborts the load.

To reduce the number of data exports, you are encouraged to use the `HIDERL` command to hide data symbols. Refer to "Hiding Data Symbols".

Note

XLs can now contain duplicate data symbols, as long as the duplicates occur in different modules. When adding modules to an XL, if another module has a data item with the same name, the second module is also added to the XL. For Data Universal exports, when replacing modules, only the *first* module found exporting that symbol is replaced.

When adding modules from a RL, a new parameter can be used to instruct the Link Editor to add only the module that exports the specified data symbol:

`DATA=`*data_name*

An indirect file can be specified for this parameter. Note that *data_name* is case-sensitive and must be a Data Universal. If *data_name* is a Data Unsat or Storage Universal, the `ADDXL` command ignores it. If `DATA=` is used without the `SHARE` option, the `ADDXL` command ignores the `DATA=` parameter.

Example

```
ADDXL FROM=TESTRL2;TO=MYXL3
ADDXL FROM=DKRL;DATA=MYTIME,index,link_time;TO=MYXL4;SHARE
```

In the first line, modules from `TESTRL2` are added to `MYXL3`. Data is not shared, so the linking behavior is the same as for non-shared global data environments. Data will not be exported nor imported in `MYXL3`.

In the second line, modules from the RL `DKRL` are added to `MYXL4`. Only modules that export `MYTIME`, `index`, and `link_time` (Data Universal exports) are added to the XL. `MYXL4` contains data exports (and imports if there are any) that other XLs and program files can access.

Manipulating XLs

The `COPYXL` Link Editor command now allows modules to be copied based on data exports (Data Universals only). This implies that the XL was built for shared data and, therefore, has data exports. The `COPYXL` command accepts a new parameter:

`DATA=`*data_name*, *data_name*...

Note that *data_name* is case-sensitive. The first module found that exports the specified data symbol is copied from the XL. If the module is not a shared global or module, the `DATA=` option is ignored.

Example

```
COPYXL FROM=MYXL4;DATA=myindex;TO=NEWXL
```

The first module found in `MYXL4` that exports `myindex` (Data Universal) is copied to `NEWXL`.

If there are duplicate Data Universal exports, it is recommended that you use the `MODULE=` or `ENTRY=` parameters to specify a specific module or a module that exports the entry point to be copied.

The `PURGEXL` Link Editor command allows modules to be purged, based on exports (Data Universals only). This implies that the XL was built for shared data and, therefore, has data exports. The `PURGEXL` command accepts a new parameter:

`DATA=`*data_name*

Note that *data_name* is case-sensitive. The first module found that exports the specified data symbol is purged from the XL. If the module is not a shared global data module, the `DATA=` option is ignored.

Example

```
PURGEXL XL=MYXL;DATA=bar
```

In this example, the first module found that exports `bar` is purged.

If there are duplicate Data Universal exports, it is recommended that you use the `MODULE=` or `ENTRY=` parameter to specify a specific module or a module that exports the entry point to be purged.

Listing Program Files and XLs

The `LISTPROG` and `LISTXL` Link Editor commands now list Data Unsats, Storage Universals, and Data Universals.

The `LISTPROG` command uses the `;DATA` option to print data symbols. The `LISTXL` command accepts a new parameter to allow the printing of modules that export a particular data symbol:

`DATA_ITEM=`*data_name*

Note that *data_name* is case-sensitive and can be either a Data Universal or Storage Universal.

Example

```
LISTXL MYXL;DATA_ITEM=foo
```

In this example, only the modules that export `foo` are listed.

There are new field values for shared global data modules in the `LISTPROG` and `LISTXL` output. The field values are:

- Under `Sym Type`, there is a new *symbol type*, `s_req`, which denotes a storage request.
- Under `Sym Scope`, there is an *unsatisfied scope*, `unsat`, which is now valid with symbol type data.
- Under `Sym Value`, the DP-relative DXRT offset is printed for Data Unsats: `dp` = *data pointer*.

For Data Universals in program files, the address of the literal or the offset from DP is printed. For Data Universals in XLs, the *best-guess* offset from DP is printed. This is a best-guess because the loader can override the definition at run time unless the symbol is hidden or the module is linked for non-shared data.

For Storage Universals, the best-guess offset from DP is printed (best-guess because the loader can override it at run time) unless the symbol is hidden or the module is linked for non-shared data.

A new field, `SHARED DATA`, is printed in the output listing to indicate if a module is a shared global data module. If `SHARED DATA` is set to `YES`, the module contains shared global data; if `SHARED DATA` is set to `NO`, the module does not contain shared global data.

All Data Universals and Storage Universals are listed before the Data Unsats. If the `;VALUE` option is specified, the symbols are sorted in each class by value. If the `;VALUE` option is not specified, the symbols are sorted in each class by name.

See “Sample Listings” for examples.

Loading a Shared Global Data Program File or XL

The loader is responsible for binding code and data imports and exports at load time. The loader will:

- Set up DP according to the initialization pointers.
- Set up LP and XRT entries according to LST import records.
- Initialize the DXRT entries with addresses at load time according to the loader fixups and LST import records.
- Implement a new binding scheme for data that allows program file global data exports to bind XL data imports and vice versa.
- Copy the data exports from the file's export list. Select and store a unique instance of the data export into the loader's Process Data Dictionary (PDD).

Note Code binding has not changed. XLs can bind code imports in a program file but program file code exports will not bind XL code imports.

Loader Binding Rules

The loader binds data symbols according to these binding rules:

- Data Universals in program files have precedence over definitions in XLs for that symbol because the precedence rule is observed from left to right and the program file is always first.
- For Storage Requests, the largest one encountered has precedence over all others, either in a program file or XL.
- Initialized data (Data Universals) take precedence over uninitialized data (Storage Requests).
- The most privileged data takes precedence.
- Memory resident data takes precedence over non-memory resident data.
- XL Data Universals are selected on a first-seen basis. If the loader loads a module that has a definition for `foo` and there is no other definition for `foo` in an already loaded module or if there is a Storage Universal definition in an

already loaded module, those definitions take precedence over all others. All imports for `foo` will bind to the one just loaded.

Violation of any of the above rules for any data symbol results in a failed load. The precedence is established for each distinct data symbol only *once* per process. Therefore, libraries loaded dynamically, through the HPGETPROCPLABEL intrinsic, are subject to the same rules as static XLs (XLs specified at link time).

Compatibility

Shared global data XLs and program files are not backward compatible and do not run on older operating systems. Even if you use RLs during the link and your `make` script has not changed, the loader will not load the program file or XL if there are any data exports or imports. To determine if a program file or XL contains shared data, use the `LISTPROG` or `LISTXL` command appropriately and check the `SHARED DATA` header.

Program files, object files, RLs, and XLs compiled with the 3-instruction compiler option are not backward compatible and do not run on older operating systems, even if they do not import or export any data.

Note that program files and XLs created on older systems can be run on the shared global data operating system. Relinking is not required for the program to run.

Note

POSIX programs must be recompiled if they contain the header files `<stdio.h>` and `<direct.h>`. If a program contains a direct or indirect reference to `__file`, it must be recompiled.

The macros `_file`, `_bufend`, and `fileno` contain references to `__file`

Linking with Compatibility Mode Files

The Link Editor does not allow users to share data when linking with compatibility mode files. For example, if `octcomp.pub.sys` is invoked and `LKSHAREDATA` is set to true, the Link Editor issues an error message and aborts the link.

Refer to “Diagnostic Messages” for the error message.

Advanced Topics

This section describes additional information on shared global data in XLs.

External Data References

An external data reference, also known as an import, is a reference to a data symbol that is not defined in the module. External data references are now allowed in program files and XLs if they are linked for shared data. These references are not resolved at link time because the Link Editor builds one load module at a time, using `LINK` or `ADDXL`. Therefore, the Link Editor does not know where the corresponding data definition, or export, is located. The data item could be located in a module in the same XL or in another XL.

Because the Link Editor cannot resolve external data references, it turns the reference into an LST data import request. It also allocates an entry in the Data Cross Reference Table (DXRT) for the unresolved data reference. The instruction used to reference the external data is changed from a direct reference to an indirect reference through the DXRT. This indirection enables the loader to locate data definitions for each data reference and initialize the DXRT with the appropriate values at run time.

External data references can be created by the Link Editor for Storage and Data symbols. When linking to form a program file, the Link Editor creates a corresponding Data Unsat record for each Storage Universal. The Link Editor creates the Data Unsat to allow other definitions or Storage symbols to override the original definition or symbol at run time. The Link Editor also creates an LST data import record and allocates an entry in the DXRT for the import. At run time, the loader locates data definitions for each data reference, including those that the Link Editor created. For Storage Universals, if a larger Storage Universal or a corresponding Data Universal is found, the loader ignores the *original* Storage Universal and binds all references to the symbol found.

When adding modules to an XL, the Link Editor creates a corresponding Data Unsat record for each Data Universal and goes through the same process used for creating imports for Storage Universals, outlined above. This allows other definitions to override the original at run time. Data Universals in program files do not have Link Editor-created Data Unsats because Data Universals in program files always take precedence over symbols in an XL. Therefore, Data Universals in the program can never be overridden at run time.

Duplicate Data

Duplicate data exports are allowed in an XL, as long as the duplicates occur in separate modules. Duplicate Data Universal exports are *not* allowed in program files and are not recommended in XLs. Duplicate Data Universals are not recommended in XLs because duplicate data exports are not allowed in HP-UX Shared Libraries. Also, the loader uses the first Data Universal it finds as the defining definition. The loader picks one of the duplicate symbols and uses it as the *single* overriding definition for all references to it.

There is a distinction between Data Universal and Storage Universal data exports because there is apt to be more duplicate Storage Universals than Data Universals. For example, Data Universal exports do not exist in Pascal object files. All declarations are considered Storage Unsats.

The XL commands that support the `DATA=` parameter (`ADDXL`, `COPYXL`, `PURGEXL`) work on Data Universals and ignore Storage Universals. The `LISTXL` command is an exception. `LISTXL DATA_ITEM=` lists all modules that export the specified Data Universal or Storage Universal, it does not *change* the XL in any way. Therefore, it is less of a risk to list every module that exports a data item than to only list one.

The `DATA=` parameter works on the *first* module it finds that exports the Data Universal symbol. Because duplicate data is legal, the Link Editor cannot predict if you want to process only one module that exports the data item or all modules that export the data item. To ensure that you are processing a specific module and not just the first one the Link Editor finds that exports a specific data symbol, use the `ENTRY=` or `MODULE=` options of the `ADDXL`, `COPYXL`, or `PURGEXL` commands. The exception is the `LISTXL` command as noted above.

The XL commands that support the `ENTRY=` parameter (`ADDXL`, `COPYXL`, `PURGEXL`) work on only one module that exports the symbol, because entry points must be unique in an XL and in program files.

Example

The following example demonstrates how the PURGEXL and LISTXL commands process an XL that contains duplicate data exports. Assume that after each PURGEXL command is executed, the process starts over with the same MYXL XL.

```
MYXL

Module DKTEST
   i data univ        /* i, foo, fee are exports */
   foo data univ
   fee s_req univ

   fum data unsat     /* fum, foo, fee, i are imports */
   foo data unsat
   fee data unsat
   i data unsat


Module DKOPEN
   fee s_req univ     /* fee is an export */

   fum data unsat     /* fum, fee are imports */
   fee data unsat


Module DKCLOSE
   foo data univ      /* foo, bar are exports */
   bar entry univ

   foo data unsat     /* foo is an import */


Module DKGET
   bar data univ      /* bar is an export */

   bar data unsat     /* bar is an import */
```

`PURGEXL XL=MYXL;DATA=foo`	Deletes module `DKTEST` from `MYXL` because `DKTEST` is the first module found that exports `foo`.
`PURGEXL XL=MYXL;DATA=fee`	Does not delete any module from `MYXL` because no module was found that exports a data universal symbol named `fee`.
`PURGEXL XL=MYXL;DATA=fum`	Does not delete any module from `MYXL` because no module was found that exports a data universal symbol named `fum`.
`PURGEXL XL=MYXL;DATA=bar`	Deletes module `DKGET` from `MYXL` because `DKGET` exports a data universal symbol named `bar`.
`PURGEXL XL=MYXL;ENTRY=bar`	Deletes module `DKCLOSE` from `MYXL` because `DKCLOSE` exports an entry symbol named `bar`.
`LISTXL XL=MYXL;DATA_ITEM=fee`	Lists modules `DKTEST` and `DKOPEN` from `MYXL` because both modules export a data symbol named `fee`.

Sample Listings

Symbol Transformation Example

```
C source file

int i;
int foo=0;
extern int bar;
...
main () {
  i++;
  foo++;
  bar++;
  ...
}

Object File
--------------------------
|Symbol| Type   | Scope   |
|-------------------------
|i     |Storage|Unsat    |
|foo   |Data   |Universal|
|bar   |Data   |Unsat    |
--------------------------

Shared Data Program File
--------------------------------------------------
|Symbol| Type   | Scope   |LST Import |LST Export |
|-------------------------------------------------|
|i     |Storage|Universal|    Yes    |    Yes    |
|foo   |Data   |Universal|    No     |    Yes    |
|bar   |Data   |Unsat    |    Yes    |    No     |
--------------------------------------------------
```

```
Shared Data XL
--------------------------------------------------
|Symbol| Type   | Scope    |LST Import |LST Export |
|-------------------------------------------------|
|i     |Storage|Universal|    Yes     |    Yes     |
|foo   |Data   |Universal|    Yes     |    Yes     |
|bar   |Data   |Unsat    |    Yes     |    No      |
--------------------------------------------------

Non-Shared Data Program File or Non-Shared Data XL
--------------------------------------------------
|Symbol| Type   | Scope    |LST Import |LST Export |
|-------------------------------------------------|
|i     |Data   |Universal|    No      |    No      |
|foo   |Data   |Universal|    No      |    No      |
|bar   |Data   |Unsat-->ERROR! Not allowed         |
--------------------------------------------------
```

Key:

`LST Export = Yes`	The symbol is externally visible to other XLs and program files and can be used to satisfy LST import requests.
`LST Export = No`	The symbol is not externally visible. This is the same behavior as for non-shared global data environments.
`LST Import = Yes`	Look for LST exports in other XLs or program files that can satisfy the import request.
`LST Import = No`	The symbol is satisfied and does not look for another definition that can override it. This is the same behavior as for non-shared global data environments.

LISTPROG Example

```
LinkEd> listprog myprog

PROGRAM        : MYPROG
CAPABILITIES   : BA, IA
NMHEAP SIZE    :
NMSTACK SIZE   :
ENTRY NAME     :
UNSAT NAME     :
PRIORITY       :
MAX PRIORITY   :
POSIX          : NO
SHARED DATA    : YES
TEXT SIZE      : 000002F8
DATA SIZE      : 0000001C
VERSION        : 85082112

Sym                  C H X P Sym    Sym         Sym      Lset
Name                         Type  Scope      Value    Name
----                 - - - - ----  -----      -----    ----
$START$              0   3 3 pri_p univ       00005004
main                 0   3 3 entry univ       00005040
$RECOVER_END         0       code  univ       000052F8
$RECOVER_START       0       code  univ       000052F8
$START$              0       code  univ       00005020
$UNWIND_END          0       code  univ       000052D8
$UNWIND_START        0       code  univ       000052A8
```

```
_start               0 H     code  univ       00005218
main                 0       code  univ       0000509C
M$6                  0       data  local   dp+00000008
foption              0       s_req univ    dp+00000018
myopt                0       data  univ    dp+00000000
ARITRAP              0       stub  ext     lp+000000A0
U_INIT_TRAPS         0       stub  ext     lp+000000C0
_exit                0       stub  ext     lp+00000160
_close               0       stub  ext     lp+00000060
_dup                 0       stub  ext     lp+00000100
_init_c_globals      0       stub  ext     lp+00000040
_init_x11_globals    0       stub  ext     lp+00000140
_open_std_file       0       stub  ext     lp+000000E0
_parse_info_string   0       stub  ext     lp+00000020
foo                  0       stub  ext     lp+00000120
printf               0       stub  ext     lp+00000080
foption              0       data  unsat   dp-00000014
mystring             0       data  unsat   dp-00000010
myvalue              0       data  unsat   dp-0000000C
```

LISTXL Example

```
listxl dkxl4

LIBRARY NAME   : DKXL4
VERSION        : 85082112
MODULE COUNT   : 2
MODULE LIMIT   : 500

MODULE NAME                          START     LENGTH
-----------                          -----     ------
HIMOMC                             00129000   0000355C
MYBAR                              00139000   0000336C

MODULE NAME    : HIMOMC
TEXT SIZE      : 00000080
DATA SIZE      : 00000008
VERSION        : 87102412
LENGTH         : 0000355C
SHARED DATA    : NO

Sym                    C H X P Sym    Sym        Sym     Lset
Name                           Type   Scope      Value    Name
----                   - - - - ----   -----      -----    ----
main                   0   3 3 entry univ      0012B000
$RECOVER_END           0       code  univ      0012B080
$RECOVER_START         0       code  univ      0012B080
$UNWIND_END            0       code  univ      0012B070
$UNWIND_START          0       code  univ      0012B060
main                   0       code  univ      0012B03C
printf                 0       stub  ext     lp+00000020
```

```
MODULE NAME    : MYBAR
TEXT SIZE      : 00000004
DATA SIZE      : 00000010
VERSION        : 87102412
LENGTH         : 0000336C
SHARED DATA    : YES

Sym                 C H X P Sym    Sym         Sym      Lset
Name                        Type  Scope       Value     Name
----                - - - - ----  -----       -----     ----
$RECOVER_END        0       code  univ        0013B000
$RECOVER_START      0       code  univ        0013B000
$UNWIND_END         0       code  univ        0013B000
$UNWIND_START       0       code  univ        0013B000
d                   0       s_req univ     dp+00000008
f                   0       data  univ     dp+00000004
fee                 0       data  univ     dp+00000000
k                   0       s_req univ     dp+0000000C
foo                 0       data  unsat    dp-0000000C
```

Diagnostic Messages

New warning and error messages returned by the Link Editor.

User Errors (1000-1499)

1167	MESSAGE	`Expected NMXL file. File "!" is a CM file (LINKERR 1167)`
	CAUSE	The Link Editor encountered a compatibility mode file when linking and shared data was enabled.
	ACTION	Make sure LKSHAREDATA is unset or set to FALSE before attempting to link with compatibility mode files.

Warning Messages (1500-1999)

1562	MESSAGE	`SYMBOL "!1" IS HIDDEN/NOT HIDDEN IN DIFFERENT MODULES. HIDING IT (LINKWARN 1562)`
	CAUSE	The Link Editor encountered a hidden and non-hidden version of the same symbol during the link.
	ACTION	Make sure all definitions in the object modules you are linking with are hidden or make sure that all definitions are not hidden. A mismatch of hidden and unhidden symbols causes the Link Editor to hide all occurrences it sees.

Language Subsystem Errors (3000-3999)

3060	MESSAGE	`REFERENCE TO "!" IN FILE "!" NEEDS 3-INSTRUCTION SEQUENCE -- USE +k OR $MORE_GLOBALS$ TO RECOMPILE (LINKERR 3060)`
	CAUSE	DXRT limit of approximately 2k was hit. There are too many data imports and/or too many imports requiring multiple DXRT slots.
	ACTION	Recompile with the `+k` option or `$MORE_GLOBALS$` pragma to compile for 3-instruction load and store sequences. The DXRT can be bigger than 2k because far away slots can be reached by using the extra instruction generated.

Glossary

Data Universal
An initialized data symbol defined in an object module that is visible (exportable) to other object modules.

Data Unsat
A data symbol that is referenced by an object module but not defined in it. These symbols are allowed in XLs and program files for shared global data.

DXRT
A Data Cross Reference Table. The table is initialized by the loader at run time with data definition addresses. All data references that need a corresponding definition has an entry in this table. This table is created by the Link Editor for shared global data support.

Export
A variable or procedure that is defined in a module and sharable with other modules outside of the one it is defined in. It can be used to satisfy references (imports).

Global Data
Data definitions, either initialized or uninitialized, that are exported to or imported from other compilation units.

Import
A reference in a module to a variable or procedure that is defined in another module. The request (import) can be satisfied by a corresponding export.

Object File, Object Module
A file, NMOBJ, produced by compilers and used as input to the Link Editor.

Process Data Dictionary

The Process Data Dictionary (PDD) is the repository of all data symbol resolution that has taken place on a process. Its main functions are to:

- Maintain information on all resolutions.
- Enforce resolution semantics by controlling the addition of new symbols. All arbitration between duplicate definitions is done by the dictionary.
- Maintain information to undo the effects of a failed load.

Shared Global Data

Data definitions, either initialized or uninitialized, that are exported to or imported from other compilation units. They are sharable with other modules if they are exportable. That means they are not marked as *hidden* and, therefore, ignored by the loader.

Storage Universal

An uninitialized data symbol created when an uninitialized data declaration is made. If no corresponding Data Universal is found for it during linking, the scope is changed from Unsat to Universal and the symbol is exportable and can be used to satisfy references (imports).