INGLENET
business solutions

# Heritage Support Package for OS/2200

IP-635

Confidential

Feb 2007

This edition applies to TIP Studio 2.5 and revision levels of TIP Studio 2.5 until otherwise indicated in a new edition. Publications can be requested from the address given below.

Inglenet Business Solutions Inc reserves the right to modify or revise this document without notice. Except where a Software Usage Agreement has been executed, no contractual obligation between Inglenet Business Solutions Inc and the recipient is either expressed or implied.

It is agreed and understood that the information contained herein is **Proprietary** and **Confidential** and that the recipient shall take all necessary precautions to ensure the confidentiality thereof.

*If you have a license agreement for TIP Studio or TIP/ix with Inglenet Business Solutions Inc, you may make copies of this documentation for internal use.* Otherwise, you may not copy or transmit this document, in whole or in part, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of Inglenet Business Solutions Inc.


Inglenet Business Solutions Inc

| | |
|---|---|
| Toll Free: | 1-800-387-9391 |
| Website: | http://www.Inglenet.com |
| Sales: | Sales@Inglenet.com |
| Help Desk: | HelpDesk@Inglenet.com |


TIP Studio, TIP/ix, and TIP/30, and are registered trade marks of Inglenet Business Solutions Inc:

This documentation occasionally makes reference to the products of other corporations. These product names may be trade marks, registered or otherwise, or service marks of these corporations. Where this is the case, they are hereby acknowledged as such by Inglenet Business Solutions Inc.

# Contents

# Heritage Support Package For Unisys 2200

## OS/2200 to TIP/ix Conversion Utilities

### Introduction

This document describes special purpose conversion utilities supplied with TIP/ix. These utilities are provided to assist users moving applications and data files from the Unisys 2200 OS/2200 environment to the TIP/ix UNIX environment.

Some of the HSP/22 utilities are used for migrating both System 80 and 2200 COBOL applications, so some of the options in this manual do not apply to the 2200 environment.

The terms copybook and COPYPROC are used as synonyms in this manual.

### Summary of Utilities

The following table summarizes the utility programs available and their purpose:

| Program | Functions |
|---|---|
| armdata | Load file (usually 9-track magnetic tape) produced by OS/2200 COPY,G or COPOUT into UNIX file system (C-ISAM compatible).<br>Uses specified COBOL COPY element(s) to assist translation of data from EBCDIC to ASCII or from OS/2200 data representation to appropriate UNIX representation. |
| armrpg | Convert RPG-II source programs into UNIX COBOL-85 source programs. This converter recognizes batch, on-line IMS and on-line TIP/1100 RPG-II programs and transforms them into appropriate COBOL-85 programs. |
| armsort | Utility to properly handle sort statements in the ECL stream. Can use both UNIX sort and Syncsort |

## UNIX Environment Variables

You should review the following UNIX environment variables:

**TMPDIR**
: Some versions of UNIX allow you to override the default directory for temporary files, /tmp, by specifying an environment variable, TMPDIR, that specifies where to put temporary files.

For a combined list of TIP/ix, HSP/80, HSP/22, and other related environment variables, browse the file $TIPROOT/scripts/arm.tipsetenv.

## Considerations

Operating in the UNIX environment generates some considerations for customers porting applications from mainframes. UNIX does not split blocks into logical records for your applications. For example, if your data has a logical record length of 80 bytes, and is blocked into 4000 byte blocks, your UNIX COBOL application by default will only see the first record from each block! This can be resolved by using the armdata utility to read the tape and create an output file with one record in each block.

Another consideration is that UNIX does not provide standard tape label processing. Again, this can be resolved by using the HSP/22 armdata utility.

# armdata - Read/Write OS/1100 Data Tapes

The armdata utility:

- loads data files to UNIX from tapes produced by OS/2200 in COPY,G format
- loads source code from OS/2200 in COPOUT format
- stores data files from UNIX to (optionally labeled) tapes suitable for OS/3 and other EBCDIC systems.

The armdata program is a file creation/deletion, backup/restore utility. You may create Indexed, Sequential or Relative files. You may load data into these empty files from an existing UNIX file or transfer an existing file to any valid UNIX file (for example: disk, diskette, tape...).

Unlike mainframes, UNIX does not provide record blocking and deblocking facilities. Fortunately, you can get around this problem by using armdata to copy blocked files from tape to an unblocked disk file.

## Before Using:

Find the device driver name for the tape drive on your system.

For example:

```
/dev/rmt/tg6 or /dev/rmt/tg6n
```

A hint for finding the device driver names is to look in the /dev directory then look in the subdirectory /rmt. You can try each one of the drivers you find here until you find the one that lights up the tape drive. You can use the UNIX dd command for this purpose.

If you want to read or produce *labeled tapes*, you must tell UNIX “do not rewind the tape upon closing.” There are two ways to do this:

Specify a device name which means “do not rewind” in the armdata command file. This device name usually ends (or begins) with the letter ‘n’. Or

Set up the TIPTAPENRWD environment variable to specify “no rewind” as described in the following section.

## Set Up Environment Variables:

If the input device name is not specified in the armdata command file, armdata accepts it from the following environment variables:

**TIPTAPE**
This device name means "rewind after closing".

**TIPTAPENRWD**
This device name means "do not rewind after closing".

If both of these variables are set, TIPTAPE overrides TIPTAPENRWD. Therefore, do not specify TIPTAPE if you want to suppress rewind.

The command file (previously called the grammar file) always overrides the environment variables:

**TIPTAPEREAD**
If TIPTAPEREAD is set to ‘F’ then armdata will use the UNIX fread command to read the input file instead of the default read command. The fread command will buffer input as opposed to reading the exact number of bytes requested. This is necessary on some UNIX systems (such as Data General) whose device drivers are limited to reading a fixed number of bytes (usually 512).

For a combined list of TIP/ix, HSP/80, HSP/22, and other related environment variables, browse the file $TIPROOT/scripts/arm.tipsetenv.

## Using COPYPROCS

If your file contains computational or packed data, it is easiest (but not required) to supply a COBOL copybook describing the data layout. If your file contains multiple record types, you will need to know the record identifier for each type to tell armdata how to convert the file.

## Types of Input

The armdata utility can process the following inputs:

| Input | Description |
|---|---|
| OS 1100/2200 | OS 1100/2200 SDF containing a data file in COPY,G format even if it contains binary and PIC 1 format data. Record sizes may expand.<br><br>OS 1100/2200 SDF of a print file in COPY,G format.<br><br>OS 1100/2200 COPOUT file format.<br><br>On the 1100 or 2200, the size of a COPY,G block is one “track” (7176 bytes of 9 bits). On UNIX, this corresponds to a blocksize of 8073 (8-bit bytes). |
| Flat ASCII | Flat ASCII files in UNIX, DOS or sequential (no record separators) format. The block size may be up to 32,768 bytes. |
| Interchange Format | Tapes created in Interchange Format. The interchange format will require a copybook if the file contains any signed numerics. The block size may be up to 32,768 bytes. |
| OS/3 | Labeled tapes created by OS/3 data utilities — even if they contain binary data. The block size may be up to 32,768 bytes.<br><br>Note: We strongly recommend creating labeled tapes because the labels contain the record size and block size – and the operating system does not lie about record sizes. |

If your file contains non-character data, you must describe the record layout. (The easy way is to provide a COBOL copybook. The hard way is to supply the offsets on the command line.) The armdata utility can read both labeled or unlabeled tapes from the 1100/2200.

**Syntax**

```
armdata [-Q] -Gfilename
```

```
armdata [-Q] -R:copyname
```

**Where:**

**-Q** Do not display or update the number of "Records restored". Specify this option if you want to redirect armdata messages to a file.

**-G** Use an armdata command file.

**filename**
The name of the input armdata command file.
Example: armdata -G/u/fred/paygram

**-R:** The armdata utility will review the specified COBOL copybook and report whether it detects fields whose definition will vary depending on the COBOL compiler. See later discussion of the "ID IS" clause.
If the "R" is followed by "S" (that is, -RS:), the report produced is a summary of one line for each input copy module.

**copyname**
The filename of the COBOL COPY module to be examined.
Example: armdata -R:paymast

Normally, you specify all parameters to **armdata** in a command file containing keywords and specifications (in ASCII). The only command line parameter expected by **armdata** is a specification of the filename where the armdata command file is located.

The armdata utility may also be driven with command line options, however this is recommended for simple commands that will be performed once. When using the option method you are not required to supply a copybook that defines the record layout, you may explicitly tell armdata where the computational and packed fields (if any) reside. The options are listed in later this section. Future enhancements to the armdata command file will not be added to the option list.

To enable armdata to process files with multiple record types, you must identify each record type to armdata via the ID IS statement. If you have multiple conditions that determine the uniqueness of each record type, you will have to separate the different types before processing. For example:

```
if field-a = 1 AND field-b < 100 then
record type = type A
```

is not supported. If necessary, write a utility to separate the records.

The TIP/ix system includes a UNIX script, $TIPROOT/scripts/armcbchk, that calls armdata with the -RS option for every file in the directory. This only makes sense for files that are COBOL copybooks. If there are more

than 24 copybooks in the directory, we recommend that you redirect the output into a separate file (for example, armcbchk > xxx)

**Execution:**

You can invoke armdata by one of three names, depending on which ISAM file system you will use:

| Name | Considerations |
|---|---|
| armdata | For C-ISAM file structures. This uses D-ISAM from Byte Designs (or Micro Focus batch created files). |
| armdmbp | For MBP-ISAM users. |
| armdmbp4 | For MBP ISAM users with files having an index partition larger than 64 megabytes (up to 256 megabytes in size) |

File definitions as specified to the TIP/ix smfile program must reflect which ISAM is in use. The default is D-ISAM. If you use MBP ISAM, you must specify mbpfcs in the FCS Server field of the file definition. Specify mbp4fcs for files loaded with armdmbp4.

A backup and subsequent restore of an indexed file reorganizes it so that the data partition is in sequential order. This reduces the size of the index partition.

The armdata utility looks in the TIP/ix catalogue for file information only when requested to do so, (via the READ TIPIX CATALOGUE statement). If the output file is not defined to TIP/ix, the output file will be created in the current directory.

## Using armdata with an armdata Command File

Following is a summary of the command file accepted by armdata. All armdata command file statements must begin in column 12 or greater.

```
BACKUP
CREATE
DELETE
RESTORE
ADD lfn [TO] TIPIX Catalogue
CASE [IS,=] {UPPER|LOWER}
TIPIX FILE [IS,=]
{INDEXED|RELATIVE|SEQUENTIAL}
BACKUP FILE [IS,=] (UNIX|
DOS|OS3|INTERCHANGE|COPYG|SDF|COPOUT|
FLAT <- default }
RECORD SIZE [IS,=] nnn
BLOCK SIZE [IS,=] nnn (block size of data
on tape)
```

```
COMPILER [IS,=] {MF|MBP}
PROCESS nnn RECORD(S)
LABEL RECORDS ARE {STANDARD | OMITTED}
TRANSLATE TO {EBCDIC|ASCII|ASCII-SIGNS}
DUPS COUNTER [IS,=] {HALFWORD|FULLWORD}
INPUT FILE [IS,=] filename
OUTPUT FILE [IS,=] filename
RAW field-name
RELAXED [SIGNED] [FILL [IS,=] {SPACES|LOW-
VALUES}]
MASKFILE file-name
SKIP nnn [RECORDS]
VOL [IS,=] xxxxxx
LBL [IS,=] yyyyyyyyyyyyyyyyy
KEY1 [IS,=] field-name
KEY2 [IS,=] field-name2 [PRIME|DUPS|NODUPS]
copybook [IS,=] COPY filename
ID IS fielda > value USE field1.
ID IS fieldb > value USE field2.
ID IS fieldc > value USE field3.
ID IS a AND b [...AND c] USE field4.
ID IS d OR e [...OR f] USE field5.
ID IS a AND b [AND c] OR d AND e [AND f]
USE field6.
ID IS (a AND b AND c) OR (d AND e AND f)
USE field6.
VALIDATE NUMERIC
EXPLODE [SIGN LEADING | TRAILING]
```

**Where:**

**BACKUP**

Take the input file which is in TIP/ix format and copy it one record at a time to the output file which is specified by the BACKUP FILE statement.

**CREATE**

Create an empty file named according to the specification for OUTPUT FILE. For indexed files, two files are created: one for the data partition and one for the index partition. When using armdata the extensions are ".dat" and ".idx" respectively.
When using armdmbp or armdmbp4 the extensions are ".DAT" and ".KEY" respectively.

**DELETE**

Delete the file specified by OUTPUT FILE.

**RESTORE**

Read the input file and write it one record at a time

to the output file.
CREATE and RESTORE may be included in the same armdata command file. The file will be created then loaded if this is the case.

**CASE [IS,=] {UPPER|LOWER}**
Translate the complete path and file name in the Label/Path field of the file record to uppercase or lowercase. The default is lowercase.

**ADD lfn [TO] TIPIX [Catalogue] [OVERwrite]**
Optional specification to add the output file specifications to the TIP/ix catalogue information (as if it had been defined via SMFILE).

**"lfn"** is the name of the file as it is known to TIP/ix. The name is forced to uppercase and added to the list of TIP/ix files with the current parameters as specified in this armdata command file (such as: key values, record size, file type etc.).

**OVERwrite**
causes the catalogue entry to be overwritten if it currently exists.
The armdata utility checks the catalogue to see if the LFN already exists.

If the LFN does not exist, armdata looks for the DEFAULT record and uses it to build LFN, if either are found and overwrite was selected, armdata updates the time and date stamp, and fills in any required fields which may have been left blank.

These hard coded default fields are as follows:

file access = read/write,
file sharing = shared;
record holding = hold for transaction;
record journal = no;
record logging = no;
load to memory = no.
If neither the LFN nor DEFAULT exists, armdata will create the system record with the aforementioned default values.

**TIPIX FILE [IS,=] {INDEXED|SEQUENTIAL|RELATIVE}**
Used with the CREATE or BACKUP command to specify which type of file to create. This specification must be supplied.

**BACKUP FILE [IS,=] {UNIX | DOS | OS3 | INTERCHANGE | COPYG | SDF | COPOUT | FLAT}**
If this statement is omitted, the default file type is FLAT.

**“FLAT”**
means a flat ASCII file containing no record separators.

**"UNIX"**
means that records are separated by a carriage return.

**"DOS"**
means that records are separated by a carriage return and a line feed.
Do not specify “DOS” or “UNIX” for input records that contain any characters of value less than a space ( < X’20’).
Instead, specify “FLAT” and use the block and record sizes to specify how the data will be read.
Writing “DOS” or “UNIX” records that contain characters less than a space is allowed, but is not recommended.

**“INTERCHANGE”**
is a standard inter computer format containing no binary data, however signed numeric data will require a mask to allow proper conversion

**“COPYG”**
an OS/1100 format in 9 bits / byte

**“SDF”**
an OS/1100 format containing 9 bits/byte

**“COPOUT”**
an OS/1100 @COPOUT,S format

**“OS3”**
implies that you are using an EBCDIC file. (You are reading it from, or creating it for OS/3).

**RECORD SIZE [IS,=] nnn**

The armdata utility checks the TIP/ix catalogue for the record size when the READ TIPIX CAT option is used. If the file is defined to TIP/ix, the record size is taken from the TIP/ix catalogue since this is the size FCS will use when accessing the file.

**nnn** is the actual size in bytes of each record in the output file.

If the input tape has standard labels, the information in the tape label is considered correct and this clause is ignored.

If a copybook is supplied, and its size differs from the label size, then armdata will prompt the user to make a decision on the actual record size.

This clause is not necessary when a copybook is supplied.

**BLOCK SIZE [IS,=] nnn**
Used with files that are blocked on tape. Also used when

the input size differs from the output size ( input/block size = x output/record size = y )

**nnn** is the block size in bytes of the data on tape.

If the input tape has standard labels, the information in the tape label is considered correct and this clause is ignored.

**COMPILER [IS,=] {MF|MBP}**

Since the Micro Focus and MBP COBOL compilers expect signed numeric data in different formats, armdata needs to know which compiler you are using so that it can convert the data correctly.

**PROCESS nnn RECORD(S)**

**nnn** is the number of records to process before termination.

This can be useful in creating sample data where only a portion of the file is transferred.

Optional parameter; default is process all records in the file.

**LABEL RECORDS ARE STANDARD | OMITTED**

Specify if the tape is labeled. The default is STANDARD (the tape has a label).

**TRANSLATE TO {EBCDIC | ASCII | ASCII-SIGNS}**

Specify the desired translation for the output file.

**TO ASCII**

to translate an EBCDIC input file to an ASCII output file.

**TO EBCDIC**

to translate an ASCII input file to an EBCDIC output file.

**TO ASCII-SIGNS**

to take input that is already in ASCII and fix unpacked signed fields so that they are in either MF or MBP format (according to the COMPILER IS parameter). This feature is useful if you use ftp to translate mainframe EBCDIC data to ASCII and transfer it to Unix. Note that in this case, the conversion to ASCII is done by ftp, not by armdata.

Optional parameter; default is no translation done (unless BACKUP FILE = OS/3 is specified).

**DUPS COUNTER [IS,=] {HALFWORD | FULLWORD}**

Set the field size where COBOL maintains the number of records which have the same value as the current key:

Use with D-ISAM (Micro Focus) files only. The default value of this optional keyword is HALFWORD.

**HALFWORD**
Allocate a two-byte binary field, limiting you to a maximum of 32767 duplicates on any key.

**FULLWORD**
Allocate a four-byte binary field.

**NOTE:** Tests reveal that contrary to earlier armdata documentation, using MF COBOL V4.x with IDXFORMAT "4" declared in the select clause does not create or accept a C-ISAM file structure capable of recognizing a FULLWORD duplicate key counter.

You can implement a FULLWORD counter, if you link the batch COBOL program with the D-ISAM library supplied with TIP/ix.

```
.bat:
cob $(FLAGS) -k $(@F).bat -o $(@F) -m
* ixfile=cixfile
* L$(TIPROOT)/lib +ldisam
```

After armdata creates a file with DUPS COUNTER IS FULLWORD, always use the OPEN I-O statement in your batch program. The OPEN OUTPUT statement causes COBOL to operate with the default HALFWORD counter size. Use the dcheck utility to show the file status.

**INPUT FILE [IS,=] filename**
Specify the filename that armdata opens for input (read).

The filename may be the full path name contained in double quotes OR just the name of the file (with or without quotes) if it is located in the current directory.

Example:

```
INPUT FILE IS "/tipix/files/datafile"
```

This clause is intended to be used to allow the specification of the input file when the input is not from a tape (for example, the data is already in a UNIX file on disk.)

If the input to armdata is a tape, you don't need to use the INPUT FILE line, but armdata expects two environment variables to be defined,

TIPTAPE
This environment variable defines the UNIX device name to be used for the tape when rewind is required.

TIPTAPENRWD

This environment variable defines the UNIX device name to be used for the tape when no rewind is required.

Example:

```
TIPTAPE="/dev/rmt/0m"
```

Example:

```
TIPTAPENRWD="/dev/rmt/0mn"
```

INPUT FILE = will override these two environment variables.

**Note:** If the first character is a dollar sign ($), then it will be interpreted as an environment variable up to the first '/'. (that is, $HOME/myfile will have $HOME expanded and /myfile appended to the end).

**OUTPUT FILE [IS,=] filename**

Specify the filename that armdata opens for output (write). filename may be the full path plus the name contained in double quotes OR just the name of the file (with or without quotes) if it is located in the current directory.

Example:

```
OUTPUT FILE IS "/tipix/files/xxxfile"
```

**Note:** If the first character is a dollar sign ($), then it will be interpreted as an environment variable (see INPUT FILE).

**RAW field-name**

Transfer the exact image for the field specified by field-name. This may be used for fields defined as alphanumeric that contain binary sensitive data. For multiple fields, include a separate RAW declarative; up to a maximum of 10.

**RELAXED [SIGNED] [FILL [IS,=] {SPACES|LOW-VALUES}]**

Ignore binary sensitive data; while continuing to write the records. This is useful if you don't have a copybook defining the record layout.

The SIGNED option checks for signed numeric data and converts the sign byte correctly (NOTE: One byte signed numerics PIC S9 will NOT be found).

A report specifying the location of each offensive byte found along with its hexadecimal value is written to the file named <filename>.rpt in the current directory.

The FILL option will overwrite all binary sensitive data with the supplied character (a space or a low-value) The FILL option will not create a report file.

When a copybook is supplied, the relaxed option will leave binary data in PIC X fields alone.

**MASKFILE file-name**

Tell armdata to write out what it thinks the record layout looks like (based on the copybook) in the file named file-name and stop processing.

**SKIP n [RECORDS]**

Specify (on a BACKUP or RESTORE function) the number of records (not blocks) to be skipped before processing begins.

Example:

```
SKIP 523
```

**VOL [IS,=] xxxxxx**

This keyword is used to specify the desired VOLume name of a labeled output tape that armdata is to create. Up to six characters may be specified to be used as the output tape volume serial number. If VOL is not specified, output tapes are not labeled!

Example:

```
VOL IS TAPE23
```

Both VOL and LBL must be specified if you want to write a labeled tape.

**LBL [IS,=] yyyyyyyyyyyyyyyyy**

This keyword is used to specify the desired LaBeL name of a labeled output tape that armdata is to create. Up to seventeen characters may be specified to be used as the output tape label name.

Example:

```
LBL IS RECREATED*PAYMAST
```

If LBL is not specified, output tapes are not labeled! Both VOL and LBL must be specified if you want to write a labeled tape.

**KEY1 [IS,=] field-name**

**KEY2 [IS,=] field-name2 [PRIME|DUPS|NODUPS]**

**...**

**KEY10 [IS,=] field-name2 [PRIME|DUPS|NODUPS]**

Specify the field names in the supplied copybook that represent the keys for the file being created. Up to 10 key specifications may be included. Keys 2 through 10 may (as shown) define whether DUPS are allowed on that key.

Default is DUPS on all non-primary keys.

The primary key (KEY1) must be NODUPS for TIP/ix files. To specify a key other than KEY1 as primary, use the directive PRIME. If you need the primary key to allow duplicates, specify the keyword DUPS after the key specification (for example, KEY2 = field-name2 PRIME DUPS).

**copybook [IS,=] COPY filename**

Specify the filename that contains the COBOL copybook defining the layout of the data records in this file. The filename may be specified as an absolute (full) file name in double quotes or as a relative file name.

Examples:

```
copybook IS COPY "/tipix/copy/PAYREC"
copybook IS COPY PAYREC
```

When using a relative file name, armdata looks for it first in the same directory as the command file specified by the -G option, then in the directories specified by the $COBCPY environment variable, and finally in the $TIPROOT/include directory.

**ID IS field > value USE field.**

Specify the conditions that armdata is to employ to determine which of several REDEFINES apply to a particular set of data.

Valid operators are ( >, >=, =, <>, <=, < ).

**IMPORTANT:** the ID IS statement(s) must be the last statements in the armdata command file and follow the copybook statement. Each ID statement must be terminated with a period. The armdata utility must evaluate all ID clauses for each record; thus, the last ID clause that evaluates TRUE controls how the record is processed.

The armdata utility does not process complex expressions. (for example, ID IS REC-STAT = “H” AND .... ) is not processed.

If you have redefined fields that contain binary data, you need to use the ID IS statement to specify under what conditions to use a certain redefines field.

To find out if your copybook contains any redefines of interest use armdata with the -R option.

Example:

```
ID IS REC-STAT = "H" USE HEADER-LAYOUT.
```

The field REC-STAT may have a subscript.

**IMPORTANT:** make sure the field variable is of the same type and length as the value part of the expression.

**ID IS [(] a AND b AND c [)] [...OR [(] d AND e AND f [)] ].**

If the logical expression contains both AND operators and OR operators, the AND operations are performed first.

**Note:** You cannot override this behaviour with parentheses.

VALIDATE NUMERIC

This command forces armdata to check all COMP-3 and unpacked numeric fields. For packed fields, armdata validates every digit in the field. For unpacked numeric fields, armdata only checks for space characters.

If invalid data is found, a warning message is issued.

EXPLODE [SIGN LEADING | TRAILING]

When migrating, you may want to "explode" your data into display format by expanding all COMP-3 and signed unpacked (PIC S9) fields. Using your input copybook definition as a guide, EXPLODE reformats packed and signed unpacked data as signed ASCII data.

If EXPLODE is specified in the command file, armdata adjusts the length and location of the keys. For this to work, the following assumptions must be met:

Keys must not be defined in a REDEFINES, and

Keys must not overlap.

If any of the above assumptions does not apply, try to find a work-around such as using the -k option.

In addition, a new copybook for the output is created. If the input copybook is named "file", the output is named "file.EXP".

Example:

```
EXPLODE
"9(5) COMP-3" to "9(5)"
Input x"12345F" to c"12345"
```

Example:

```
 EXPLODE SIGN LEADING
"S9(5) COMP-3" to a "S9(5) SIGN LEADING
SEPARATE"
Input x"12345c" to c"+12345"
"S9(2)" to a "S9(2) SIGN LEADING SEPARATE"
Micro Focus: Input x"3278" to c"-28"
MBP: Input x"3248" to c"-28"
```

The EXPLODE command saves you from having to write programs to explode each record layout.

### Supported ID IS Literal Types

The following table shows the supported literal types for the various field types processed in the ID IS statement.

| Field Type | Literal Type |
|---|---|
| Group / Alphanumeric | String, Numeric, Hexadecimal<br>All literals will be left aligned for comparison to the corresponding field. |
| Binary (Computational) | Numeric, Hexadecimal, Octal<br>These literals are stored into a full word, which may contain a maximum value of 2 to the 31st power. Therefore the Numeric literal may not exceed 9 digits in length, the Hex literal may not exceed 8 characters and the Octal literal may not exceed 10 digits. |
| Numeric Packed | Numeric |
| Numeric Unpacked | Numeric |

### OS 1100/2200 Specific:

When reading a tape created by COPY,G armdata will input blocks of size 8073 bytes. This is the size of a COPY,G block in 8 bit bytes.

When reading a SDF file, armdata knows the input record size, however, these records are stored in an area that is rounded up to the nearest word boundary. Records will be output according to the largest record size (supplied in the SDF header) OR the output record size will be determined from the supplied COBOL copybook.

PICTURE 1 - The 2200 COBOL compiler supports 1 bit binary fields via this clause. armdata recognizes these fields and converts them to 1 byte for each bit defined. If the bit is on, the field value is a character 1, else a character 0. This is done since PIC 1 is a non-standard extension the UNIX COBOL compilers do not support.

### Merging Files:

If you have a file to create on UNIX which arrives in two or more separate parts, perhaps because it is very large, .merging the multiple pieces can be done as follows.

To merge indexed files, load part one using armdata, then load the second part using armdata and you will get a prompt asking if you want to Overwrite, Merge or Cancel. When you select the Merge option, armdata will combine the two files for you, updating any duplicate records with the record from the second part (or latest version if more than 2 parts).

To merge sequential files, copy the files to UNIX (use the UNIX dd command if you have your files on a 9 track tape) then use the UNIX cat

command to combine the file parts into one piece. Make sure to combine them in the order they belong, with sequential files, order is important! You may now use armdata to convert the file properly. (that is EBCDIC to ASCII and binary, packed number and signed numeric conversions). Example cat command: cat file1 file2 > file3, this combines file1 and file2 into file3.

To merge a relative file, follow the same steps as for a sequential file.

armdata does handle multi-volume 9 track tapes.

### Input/Output files of Different sizes:

If you wish to expand a record upon output; RECORD SIZE = nnn will dictate the input record size and copybook IS COPY copy bookname will dictate the output record size. Expanded records will be filled with low values. (This does not pertain to COPY,G format)

### armdata Command File Example

The following example creates an indexed file named /tipix/tipfiles/insfile. The file has two keys and a TIP/ix catalog entry for the SAMPFILE file.

```
CREATE
RESTORE
TIPIX FILE IS INDEXED
ADD sampfile TO TIPIX CATALOGUE
TRANSLATE TO ASCII
INPUT FILE = sampin
OUTPUT FILE IS "/tipix/tipfiles/insfile"
RECORD SIZE = 61
COMPILER IS MF
KEY1 IS INS-KEY
KEY2 IS INS-KEY-2
copybook IS COPY "sampcopy"
ID IS INS-IDENTIFIER = "V" USE INSURANCE-01.
ID IS INS-IDENTIFIER = "A" USE INSURANCE-02.
ID IS INS-IDENTIFIER = "S" USE INSURANCE-03.
ID IS INS-IDENTIFIER = "R" USE INSURANCE-04.
```

The ID IS clauses specify when to use a particular (REDEFINES) definition of a field that has several formats.

The COPY module referred to in the above example armdata command file might look like the following:

```
*-------------------------------------------------------------*
05 INSURANCE.
10 INSURANCE-00.
15 INS-STATUT PIC X.
15 INS-KEY.
20 INS-NUM PIC S9(9) COMP-3.
20 INS-IDENTIFIER PIC X.
20 INS-NO-CONS PIC S9(5) COMP-3.
15 INS-KEY-2.
20 INS-CERTIFICATE PIC S9(9) COMP-3.
20 INS-IDENTIFIER-2 PIC X.
15 INS-KEY-ACCESS PIC X(30).
*-------------------------------------------------------------*
10 INSURANCE-01.
```

```
15 INS-IDENTIFICATION PIC X.
15 FILLER PIC X(14).
*-------------------------------------------------------------*
10 INSURANCE-02 REDEFINES INSURANCE-01.
15 INS-TERM PIC X.
15 INS-PROBLEM PIC X.
15 INS-LIFE-ASSUR PIC X.
15 INS-SALARY PIC X.
15 INS-DENTAL-PLAN PIC X.
15 INS-FAMILY-LIFE PIC X.
15 INS-AUTO PIC X.
15 FILLER PIC X(8).
*-------------------------------------------------------------*
10 INSURANCE-03 REDEFINES INSURANCE-01.
15 INS-NUM-CLAIMS PIC 9(2).
15 INS-STATUS-CODE PIC X(6).
15 FILLER PIC X(7).
*-------------------------------------------------------------*
10 INSURANCE-04 REDEFINES INSURANCE-01.
15 INS-STATE PIC 9.
15 FILLER PIC X(14).
```

The following is an example of the output from armdata:

### Using armdata from the command line

The armdata utility recognizes the command line options described in the following table. In some situations, it may be easier to run armdata via command line parameters rather than taking the time to create a separate armdata command file (as described in the previous sections).

| Option | Function |
|---|---|
| -a*name* | Add definition to TIP/ix control file. "Name" is the TIP/ix logical file name defined in the utility SMFILE. This name could be different from the physical file name. |
| -b | Backup -fi to -fo |
| -B*location*:*length*:*occurrence* | Specifies binary data field, where |
| location | The zero relative offset of the field into the record. |
| length | The number of bytes the field occupies. |
| occurrence | The number of times this field is repeated (numeric table.) If the field occurs once, you may omit this parameter. |
| -c | Create file named by -fo |
| -C*fname* | The supplied COBOL copybook is located in fname. fname is a COBOL copybook describing the data in each record of the file being converted. fname cannot have an extension. |
| -d | Delete file named by -fo. |
| -e | Backup file is in EBCDIC (translate to ASCII). File to be restored is in ASCII (translate to EBCDIC). |
| -fi | Input file for backup/restore functions |
| -F*fname* | File to place output from parsing the COBOL header file named in the -C option. This is optional, but is useful in creating a script file for further manipulation to |

be used as input for armdata. fname can not have an extension.

-fo Output file

-h Display help information

-i Indexed file operation (create)

-I *offset*EQ*string*

Specify processing using IF conditions:

offset Zero relative offset of the field in question (numeric value, for example 105).

EQ One of ( EQ, NE, LT, GT, GE or LE ) operator used when comparing the field at location to the string (comparison is for length of string).

string The value to search for (alphanumeric, for example, “RED” or “0009”).

IF conditions are followed by mask(s). See Example 4. IF conditions are useful in processing files that contain records of different layouts. In this case there would be an overriding set of masks for each IF condition met.

- Create a relative (or line sequential) file as output. These are flat sequential files that
j terminate with X'0A'.

-k#:*n*,*m*

-k#:*fieldname*

Key information for indexed files:

# key number 1-9

n key location

m key length

fieldname is the COBOL field name of the key. You MUST supply a COBOL copybook that contains this fieldname by using the -C option!

-P*location*:*length*:*occurrence*

Specifies packed data field:

location The zero relative offset of the field into the record.

length The number of bytes the field occupies.

occurrence The number of times this field is repeated (numeric table.) If the field occurs once, you may omit this parameter.

-q Create a relative (or line sequential) file as output. These are flat sequential files that terminate with X'0A'.

-Q Do not display or update the number of "Records restored". Specify this option if you want to redirect **armdata** messages to a file.

-r Restore -fi to -fo

-R[S] Report mode. Interrogate the supplied COBOL copybook and send a report to the standard output device. The report indicates whether there are fields that need special attention (binary sensitive) when converting the data.
S (summary) - produce the same information as the plain report mode (R) but reduce the output to one line per copybook.

NOTE: Inglenet Business Solutions provides a UNIX script called armcbchk that calls **armdata** with the -RS option for every file in the directory. This only makes sense for files that are COBOL copybooks. If there are more than 24 copybooks in the directory, you would be wise to redirect the output into a separate file, for example:
armcbchk > filename.

-s Record size of output data file. If a copybook is supplied, the output record size is obtained from it.

-t Backup file is OS/3 created tape (assumed to be EBCDIC)

-u Backup file is UNIX format

-U*location*:*length*:*occurrence*

Specifies unpacked signed data field:

location The zero relative offset of the field into the record.

length The number of bytes the field occupies.

occurrence The number of times this field is repeated (numeric table.) If the field occurs once, you may omit this parameter.

Since the Micro Focus and MBP COBOL compilers store unpacked signed data differently, you must specify which compiler you will be using. See the -MF and -MBP options for more details.

-v Block size of backup file

-x Display status and key information for file -fo

-y# Specify (on BACKUP or RESTORE operations) the number of records to skip before processing begins. Equivalent to the SKIP statement in the armdata command file.

-z# Specify the cutoff point, where # is the number of records to backup or restore. Could be used to load the first few records of a file to create sample data.

**Examples**

The recommended method is to use an armdata command file as shown in Example 4.

**Example 1**

```
armdata -re -fi/dev/rct0 -fo/u/tipix/tipfiles/newfile \
-P24:4 -B32:2 -U:50:4:6
or
armdmbp -re -fi/dev/rct0 -fo/u/tipix/tipfiles/newfile \
-P24:4 -B32:2 -U:50:4:6
```

This example will "restore" or copy the file from cartridge tape to the file called newfile in the tipix file directory. The data located at offset 24 will be treated as packed data for a length of 4 bytes.

The data at offset 32 will be treated as binary data for a length of 2 bytes. The data at offset 50 will be treated as unpacked signed for 4 bytes occurring 6 times. The rest of the data is converted from EBCDIC to ASCII.

The above example assumes "newfile" already exists. You may create the ISAM file shell and load it with data all in one step if you wish.

**Example 2**

```
armdata -cire -fi/dev/rct0 -fo/u/tipix/tipfiles/newfile \
-k1:0,10 -P24:4 -B32:2 -U:50:4:6 -s120
or
armdmbp -cire -fi/dev/rct0 -fo/u/tipix/tipfiles/newfile \
-k1:0,10 -P24:4 -B32:2 -U:50:4:6 -s120
```

This example accomplishes the same result as Example 1; and, in addition, it creates the ISAM shell with the primary key at location 0 for a length of 10 bytes.

**Example 3**

```
armdata -C/tipix/include/CDA -F/u/fred/filemask
or
armdmbp -C/tipix/include/CDA -F/u/fred/filemask
```

This example interprets the header file CDA in the /tipix/include directory and places its output in the file /u/fred/filemask. The output is of the form -P24:4 -B32:2 -U50:4:6, which can be used as an input mask for armdata, armdmbp or armdmbp4.

**Example 4**

OPTION METHOD

```
armdmbp -cire -fixxxfile -fotipfile -k1:0,10 -k2:10,5 \
-MBP -s120 -P24:4 -B32:2 -U:50:4:6 -I99EQ01 -B30:2 \
-I99LT10 -P100:4:2
```

ARMDATA COMMAND FILE METHOD

```
armdmbp -Gcmdfile
```

The armdata command file to perform the above commands would contain the following:

```
CREATE
RESTORE
TIPIX FILE IS INDEXED
TRANSLATE TO ASCII
INPUT FILE IS xxxfile
OUTPUT FILE IS tipfile
COMPILER IS MBP
KEY1 IS key1-field-name
KEY2 IS key2-field-name
copybook IS COPY "copy bookname"
ID IS rec-identifier = 1 USE group-item-1.
ID IS rec-identifier < 10 USE group-item-2.
```

This example reads from the file named xxxfile in the current directory and writes the converted record to the MBP-ISAM file named tipfile in the current directory.

Another possibility, with an armdata command file, is to tell TIP/ix get the real path from the TIP/ix catalogue:

```
READ TIPIX CATALOGUE
```

You may use the environment variables TIPTAPE and TIPTAPENRWD instead of the INPUT FILE = statement.

tipfile will be created with two keys as specified. The data from xxxfile will be converted from EBCDIC to ASCII and each record will be processed according to the supplied copybook (armdata command file method) or with the -P24:4 -B32:2 -U:50:4:6 mask (option method).

In addition, if the record has a "01" at location 99, -B30:2 will be processed. If the record has a value of less than "10" at location 99, then -P100:4:2 will be processed. Using the armdata command method, location 99 will be referenced by a field name; rec-identifier in this case and you must specify which part of the copybook to use when the condition is true. Usually certain areas of the file are redefined, each definition pertaining to a different record type.

**Example 5**

You may read several files from a tape by creating a file containing multiple armdata statements and executing that file under a shell.

```
armdata -cit -fi/dev/rmt/0mn -fopayroll -k1:0,10 -s220
armdata -cit -fi/dev/rmt/0mn -foiven -k1:10,5 -s160
```

The above example will read two files from the tape drive and create two indexed files: "payroll" and "inven".

The device driver used (/dev/rmt/0mn) will vary from computer to computer. When you have multiple files on one tape you need to use a device driver that will NOT rewind after the end-of-file marker is read. In the above example the "n" means "no rewind".

(To execute the script, type sh filename.)

## Transfer Source Files from 1100/2200

To transfer source programs from the 1100/2200 to UNIX you must first copy the source files to tape in @COPOUT,S format.

```
@RUN ... . Batch job
@ASG,TJ TAPEOUT.,U9S . 9-Track unlabeled tape
@ASG,A some-source-lib. . Assign source library
@PRT,TL some-source-lib. . Print table of contents
@COPOUT,S some-source-lib.,TAPEOUT. . Copy to tape
@FREE some-source-lib. . Free library
@MARK TAPEOUT. . Write tape mark
@ASG,A some-proc-lib. . Assign proc library
@PRT,TL some-proc-lib. . Print table of contents
@COPOUT,S some-proc-lib.,TAPEOUT.
```

```
@FREE some-proc-lib.
@MARK TAPEOUT.
@FREE TAPEOUT. . Free tape drive
@FIN . End of job
```

If you are sending the tapes to Inglenet for conversion, be sure to enclose the printed listings of the tables of contents.

### Example: Control with Command Files

To load the source files onto UNIX:

Create command files. For example: specific1 and specific2:

```
CREATE
RESTORE
BACKUP FILE IS COPOUT
INPUT FILE="/dev/rmt/0mn"
OUTPUT FILE"=/u/spearns/source"


CREATE
RESTORE
BACKUP FILE IS COPOUT
INPUT FILE="/dev/rmt/0mn"
OUTPUT FILE"=/u/spearns/proc"
```

Run armdata with each command file:

```
armdata -Gspecific1
armdata -Gspecific2
```

### Example: Control with Environment Variable

To load the sources (from the first file on the tape) onto UNIX:

Export TIPHOMEDATA to the desired output directory. For example:

```
export TIPHOMEDATA=/u/spearns/source
```

Create a command file, named generic, which specifies a NO REWIND tape drive:

```
CREATE
RESTORE
INPUT FILE="/dev/rmt/0mn"
BACKUP FILE IS COPOUT
```

Run armdata:

```
armdata -Ggeneric
```

On output, every source element becomes a UNIX file.

To load the procs (from the second file on the tape) onto UNIX:

Export TIPHOMEDATA to the desired output directory. For example:

```
export TIPHOMEDATA=/u/spearns/proc
```

Run armdata again. Because the NO REWIND drive was specified, the second file is read:

```
armdata -Ggeneric
```

### Additional Considerations

Logically deleted records which have a record control byte will be converted correctly in the following cases:

- the delete flag has its own byte (not part of another field) AND its value is any displayable character or high or low values ( =X”FF” or =X”00” )
- the delete flag is part of a binary or packed data field AND its value is not in the displayable character set (includes high and low values)

The above assumes you do not alter the values for =X”FF” and =X”00” in your EBCDIC to ASCII translate table in case of OS/3 migration. You may elect to simply not copy deleted records to the tape when creating it.

The delete flag will be lost if it contains a displayable character in a binary or packed data field OR a non-displayable character in an alphanumeric field.

The armdata utility can also use the translation tables generated by the TIP/ix ebcasc utility. For more information, please refer to the ARP-617-TIP/ix Utilities manual.

### Alternatives to armdata

Once you have your data either on a tape drive connected to your UNIX system or directly on disk, you can write a COBOL program to read the tape or disk file. This COBOL program would have to know the layout of the record to convert computational and packed fields properly.

Coming from OS/2200 where data is stored in 9 bit bytes (ASCII), you will usually need armdata to convert your data properly. However, if your data is all character and displayable numeric, you can either:

- Use any FTP product.
- Create an interchange tape.

Coming from OS/3, the data must always be converted from EBCDIC to ASCII.

### D-ISAM Error Codes

The following table is from the D-ISAM file system documentation, a product of Byte Designs Inc. that is included in TIP/ix. armdata intercepts and interprets some common errors, but this table provides a complete list of the possible errors that can be returned by D-ISAM.

The error message text depends on the application program (such as armdata), but generally is of the form: "Error ### accessing file" (### represents an error number greater than 99. Errors less than 100 generally emanate from the UNIX system and can be found in /usr/include/sys/errno.h.

| Error | Description |
|---|---|
| 100 | An attempt was made to (re)write a duplicate where duplicates are prohibited, or an attempt was made to REWRITE(F) where the primary key permitted duplicates. |
| 101 | The fd parameter does not reference an opened file. |
| 102 | One of the arguments has a value with no defined meaning. |
| 103 | The values of key are not valid. |
| 104 | All ISAM file descriptors are used, you cannot open any more files |
| 105 | The ISAM file is corrupted, it must be repaired with DCHECK. |
| 106 | Exclusive access to the file is not possible. |
| 107 | Another process has a read-only lock on the requested record. |
| 108 | The value of key has already been established as a key. |
| 109 | The requested function may not be performed on the primary key, as requested. |
| 110 | The beginning or end of the file has already been reached. |
| 111 | No record was found to match your request. |
| 112 | There is no "current" record set at this time. |
| 113 | The file has been exclusively locked by another process, or if trying to establish an exclusive lock, another process is using the file. |
| 114 | The name given for the file is too long or contains unacceptable characters. |
| 115 | The lock file cannot be created. Presently not used by D-ISAM. |
| 116 | malloc() cannot allocate the request. Usually means out of memory, but possibly the allocation list is corrupted. |

# armrpg - RPG-II to COBOL Converter

The armrpg utility converts RPG II source code to COBOL-85 source. Batch RPG, online IMS RPG, and TIP/1100 RPG programs may be converted to COBOL source for use with the Micro Focus or MBP compilers.

It is recommended that the RPG code is tested for clean compilation on the mainframe before going through the conversion process.

## Environment Variables

| Variable | Description |
|---|---|
| TIPRPGLDA | See Support for Local Data Areas later in this chapter. |
| TIPRPGLPP | Specify the default value of Lines-per-Page, to be used if the RPG program does not contain L-Spec cards.<br>If neither L-Spec cards nor the TIPRPGLPP environment variable are given, the default is 57. |

For a combined list of TIP/ix, HSP/80, HSP/22, and other related environment variables, browse the file $TIPROOT/scripts/arm.tipsetenv.

### Syntax

```
armrpg [options] file [... filen]
```

**Where:**

file

The name(s) of the RPG file(s) you wish to convert to COBOL 85 (with or without the extension). The armrpg utility creates a COBOL source file with the same name as your original filename. The extension depends on the type of the source program:

| Extension | Converted From |
|---|---|
| .bat | batch RPG programs. |
| .cbl | on-line TIP/1100 RPG programs. |
| .ims | on-line IMS/1100 RPG programs. |

The following options are available:

**-a** Force a full screen transmit. This is required for UNISYS-style "WorkStation" programs.

**-b** Remove all BLOCK CONTAINS clauses from the output COBOL.

**-c** When you specify -c, but not -os3, armrpg assigns the RPG program control stream reader (CTLRDR)

to standard input instead of a disk file. This permits the RPG program to "read" control stream card images from the shell script. This can simplify the creation of shell scripts for RPG programs that do not have extensive input parameters.

**-d** Insert runtime COBOL debugging statements in output source code.

**-E[X]** Specify the letter to substitute for the hyphen in the LFD name. The default is uppercase X. Do not specify a lowercase letter.
If specified, this option must be by itself or as the last option in a string because it expects a character following it. For example:

```
armrpg -dEZ
```

The mechanism used depends on which COBOL compiler you are using:

Micro Focus
Adds the keyword 'EXTERNAL' to the ASSIGN clause.
When you use armrpg -EX with Micro Focus, you must use armjcl with at least the following options:
armecl -EX

MBP Place the ASSIGN file name inside quotes. This tells MBP to use the value inside the quotes as the name of an environment variable that contains the complete file path. MBP will prefix V_ to the name of the environment variable.

Example:
The LFD name is FR-ED. armrpg changes this to FRXED. At run time, the application gets the filename from the V_FRXED environment variable.
When you use armrpg -EX with MBP, you must use armecl with at least the following options:

```
armecl -V -EX
```

Reminder: You must specify the compiler.
Either: select the -mf or -mbp option, or set the COBOL environment variable to mf or mbp.
If neither -F nor -E is specified, armrpg will create a file-assign table (in the translated COBOL code) to define the location of the specified LFD. For details, see Logical to Physical Filename Resolution in the armcob chapter.

**-foutfile**
Specify output filename. This overrides the defaults: (filename.bat, filename.ims, filename.cbl)

**-bgrpname**
is the group name for the screen formats. It is assumed that all screen formats are in the same group.

**-mf**
The target compiler is Micro Focus COBOL. This is the default.

**-mbp**
The target compiler is MBP COBOL

**-os3**
The target compiler is OS/3 COBOL-85.

**-q** Run in quiet mode. (Do not display informational messages.)

**-r** (Was -s.) Change COBOL statements in the FILE DEFINITION section from “ORGANIZATION SEQUENTIAL” to “RELATIVE”.

**-w[e]** Warn of potential character set dependencies. (for example, the RPG MHHZO verb may fail on the ASCII character set).
The ‘e’ option causes the converter to place the string “INTENTIONAL ERROR” in the COBOL source instead of displaying an error message on the screen (for example, “SORTA unsupported”).

**-IBM3** Produce code to support the IBM System/3 mode related to handling skip channel numbers. For example, “skip before channel value = 49” means “skip to output line number 49 before printing the next line”.

**-ibm** Cause armrpg to use one more character for file names (UNISYS can have 7 characters while IBM can have 8 characters in a file name).

**-u** Support UPSI bytes.

**-7** Interpret any RPG tables that start in column 7 as if they had started in column 1.
RPG expects tables to start in column 1. By default, the fse editor uses columns 1-6 for sequence numbers. This option tells armrpg to expect RPG tables to start in column 7.

## Additional considerations

The armrpg utility does not handle WORKSTATION RPG programs (batch RPG programs with screen IO embedded in them). The conversion utility will process the code, but a manual modification process will be required.

With the TIP/30 product a conversion utility exists (transaction name is RPGCNV) to convert workstation RPG programs to TIP/30 RPG programs. Perform this step first on the System/80 TIP/30 system, then

proceed to run the resulting RPG program through armrpg. This creates an on-line TIP/ix COBOL program.

### Support for Data Structures

Data structures declared in RPG are implemented as a series of data entries using REDEFINES clause. If a field appears as a data structure name or as a data structure sub-field name, the physical space reserved for that field is under these REDEFINES clauses, regardless of where the field was defined.

armrpg does not support the reorganization of fields in input records.

### Support for Local Data Area

The armrpg utility uses a library function, tiplda, for getting and putting the local data area when the program starts and ends, respectively. The module tiplda.o should be in libbat.a, libims.a, or libtip.a for compiling the COBOL programs properly.

If the converted program is started:

without ECL
: The local data area is implemented as a data file in the user's home directory. An environment variable TIPRPGLDA can be used to define a unique local data area for the user. Unlike that in RPG, it is the user's responsibility to make sure that the local data area is unique for each job or each login session.

  For example, the user can define TIPRPGLDA=$$ (and export it) to use the process number of current process. Since the process number is unique, so does the local data area. The name of the data file is lda.data$TIPRPGLDA.

with ECL
: A temporary file, $TIPCOM/lda.data, is created for each job and used as an LDA.

## armsort - Mainframe compatible sort

The purpose of the armsort utility is to provide some of the capabilities of the 2200 SORT utility. It is usually used in conjunction with the armecl utility which outputs a shell script, to execute armsort when it encounters a sort statement.

### Unsupported Keywords

Some SORT-1100 keywords do not apply to UNIX. Others just haven't been implemented yet. Customer requirements will guide future development efforts.

Currently, armsort ignores the following keywords:

| ESTIMATE | RETRIEVE | TAG |
|---|---|---|
| REELS | TAGFILE | EQUIP |
| ISKEYW | JOIN | LABEL |
| MSKEYW | PREP | PRESERVE |
| SPLIT | DATA | KEYW |
| DRUM | FAST | TAPES |
| ALLOW | CHECK | CHECKSUM |
| CKPT | CONSOLE | DELCOM |
| DELLOG | FILEID | LIMDRM |
| LIMFST | LOG | MOVE |
| NUMREC | OWN | REJECT |
| SEQ | SEQA | SEQC |

## Environment Variables

**TIPSORTWORK** Can be used to specify the name (and thus the Directory Location) of a temporary tag file used for the UNIX sort procedure. If this variable is not defined, a normal UNIX temporary file (in /tmp then /var/tmp directory) is used.

**TIPLOCKWAIT** Controls how long armsort will wait for a file.

| Value | Action |
|---|---|
| Unset | No waiting. If the file is locked, **armsort** will abort with Exit-Status of 1. |
| Zero or negative | **armsort** will try to re-access a locked file every 60 seconds. |
| Positive *n* | **armsort** will try every *n* seconds. |

**TIPHEAPS** Set to **Y** to use the in-memory heap-sort algorithm in place of the Syncsort and the UNIX sort command. Tests have shown this method is faster.

**TIPSYNCS** When the TIPSYNCS environment variable is defined (set to anything), **armsort** uses Syncsort instead of the standard UNIX

sort command.

| | |
|---|---|
| **TIPTYPEIN** and **TIPTYPEOUT** (*formerly without a TIP-prefix)* | Can be used to change the *default type* of the Input and/or Output files (respectively) from ISAM to either ASCII or DAM. That is, if either variable is set to "ASCII" or "dam", the corresponding (input or output) file would have that *type* instead of the default ISAM. |
| **TIPCENTURY** | If this variable is defined and numeric, then the value is used for the century break year. The default value for the century break year is 50. |
| **TIPUDT** | As UNIX does ***not*** examine tape labels to verify that the tape actually mounted is the one that was requested, you might want to *simulate labeled tape files on disk.*<br>Set the TIPUDT environment variable to the directory where you want to simulate a tape file.<br>set TIPUDT=/home/ian/tape |
| **TIPDEBUGALL=y** | Print a sort diagnostics report. This has the same effect as coding DEBUG=ALL in the sort command lines. |

For a combined list of TIP/ix, HSP/80, HSP/22, and other related environment variables, browse the file $TIPROOT/scripts/arm.tipsetenv.

**Syntax**

```
armsort -c file [ -d | -3 | -O ] [ -f ] [ -
t'c' ]
armsort [ -d | -3 ] [ -f ] [ -t'c' ] <<
'endstring'
statements
endstring
```

**Where:**

**-c file** The named "file" holds the processing statements. If this option is not specified, the standard input stream is read for the statements.
To read the processing statements from within a script, use a "here" file as illustrated in examples shown later in this section.

**-d** The called function is OS/3 DATA/MILOAD and the processing statements will be in that format.
This option permits file disk-to-disk copy, creating ISAM files from sequential input or creating a sequential file from ISAM input.
The UCP, UDD, UDP, UTP, SEL, SELOR, SELAND, DEL, DELOR, and DELAND statements are accepted.
For example, armsort -d (in the converted ECL-Script) will copy the contents of $INPIT1 to $PRINTR, $CARD1 to $PRINTR, and $TAPEIN to $PRINTR as the result of UDP, UCP and UTP commands respectively while recognizing

all the three forms of OC, OX & OB.
Input file(s) is defined by the environment variable(s) INPUT1, INPUT2, etc. The output file is defined by the environment variable OUTPUT1.

**-3** The called function is OS/3 SORT3 and the 'processing statements' will be in that format.
If the Input file is in sequential format, armsort can not determine the record size. Columns 35 through 38 of the "H" (header) card may be used to specify the input record size. If this is not specified then the input record size is assumed to be the same as the output record size.
The input file is defined by the environment variable INPUT (for one file), or INPUT1, INPUT2, (if there are several files).
Output file is defined by the environment variable OUTPUT.

**-O** Use OS/1100-SORT command format. Otherwise, normal SORT Commands are expected.
When using the OS/1100-Sort function of ARMSORT, the following two keywords can be used:

SKIPL=<no.>
for specifying the no. of records to be skipped from the beginning of Input-File(s).
RECNUM=<no.>
for specifying the no. of records to be sorted from Input-File(s).
Note : For detailed description, please see the OS/1100-Sort Manual

**-f** Force lowercase letters to be treated like uppercase when sorting .

**-s** Use Syncsort instead of the UNIX sort utility. If the environment variable TIPSYNCS is set, this option becomes the default.

**-t'c'** Specify the character to use to separate fields in the temporary tag file created for sorting.

**endstring**
A string to mark the end of the processing statements. "/*" is often used.

**statements**
Processing statements that are understood by the sort utility you use.
For example
```
SORT FIELDS=(2,18,CH,A)
```
If you are using Syncsort, you can pass a command line option to the Syncsort program:

sscard option
For example:
```
sscard /memory 15 megabytes
```

If options -d , -3 and -O are all missing, the SORT function is assumed and the processing commands will be in that format.

In this case, all the SORT OPTION parameters are ignored. Many of the other parameters are accepted, but since they are irrelevant on UNIX they are ignored as well. The sort field and record size information are used.

- Input files are defined by the environment variables SORTIN1, SORTIN2 , ... .
- Output file is defined by the environment variable SORTOUT.

## Using armsort

### EXTENDing the Output Files

To make armsort to add the input records to the output file (instead of initializing it), define the TIPEXTEND environment variable (TIPEXTEND=Y;export TIPEXTEND).

After executing armsort, you must unset this environment variable to avoid having this behavior applied to any following armsort commands.

If the EXTEND keyword is used with the @USE in a ECL file, the armexec function (in the converted ECL file) sets and unsets this variable.

### EXTENDing the Output Files

To make armsort to add the input records to the output file (instead of initializing it), define the TIPEXTEND environment variable (TIPEXTEND=Y;export TIPEXTEND).

After executing armsort, you must unset this environment variable to avoid having this behavior applied to any following armsort commands.

If the EXTEND keyword is used with the @USE in a ECL file, the armexec function (in the converted ECL file) sets and unsets this variable.

### Specifying Record Size for SDF Input Files From OS1100/2200

armsort is able to figure out the record size of input file except SDF files. In this case, armsort cannot find out record size directly from the structure of the file. Therefore, a new parameter, RSZIN has been added to the parameter set for OS1100/2200 files. This parameter specifies the record size of input file explicitly, thus allow users to handle the cases in which the record size of output file is different from the record size of SDF input file. Note that, if the input file is an ISAM file, armsort should be able to find out the record size itself without the RSZIN parameter.

**Example:**

```
SORTIN1=nmbrs; export SORTIN1
SORTOUT=nmbrs.srtd; export SORTOUT
rm -f $SORTOUT $SORTOUT.*
armsort -O -t':' << end-of
FILEOUT=SORTOUT BLOCK=27
FILEIN=SORTIN1 MODE=SDF BLOCK=27
DEBUG=ALL
RECORD=10
BIAS=1069
LINKSZ=1069
RSZIN=10
RSZ=20
KEY=1,7,R,$2
DISKS=3
end-of
```

In this example, the record size of SDF input file is specified as 10 bytes while the record size of output file is 20.

### Additional Debugging Information

When armsort runs it normally writes minimal information to the armecl system log, $TIPHOMEDATA/log/log.ccyymmdd. It only tells you that it ran, and the input and output record counts.

To produce a more detailed audit trail, specify the DEBUG=ALL parameter. This provides: input and output record counts, the full path of input and output files, record size, and the location of keys.

Some users find it clutters up the log, but we like it because:

- there is not much overhead, and
- this information can be very useful when tracking down a problem.

### Extension of MODE for OS1100/2200 SORT

In OS1100/2200 SORT, MODE can be either SDF or MSAM. armsort, however, accepts another, UNIX for specifying plain UNIX files for FILEIN, FILESIN, FILEOUT. MODE=UNIX can also be used for global MODE.

### Alternatives to armsort

The UNIX sort command can be used for sorting records if records are in ASCII code and in a UNIX text file. Commercial sorting packages are available from other vendors, and you may incorporate them into your systems.

# Supported OS/2200 APIs

## Common Issues

### Supported 1100 APIs

This chapter outlines the 1100/2200-API functions that TIP/ix 2.2 supports. There is a common section, and a section for each set of APIs.

Common Issues

This section describes the issues that are common to the entire 1100/2200-API. Read the sections on individual APIs for other anomalies.

### Function Names

With TIP/ix, any dollar sign '$' in a function name must be replaced with an underscore '_'. Thus a DPS function D$FuncName must be invoked as D_FuncName in the TIP/ix environment.

Why? Because Micro Focus COBOL does not allow a '$' sign in function names.

### Size of Variables

The 1100/2200 supports PIC 9(10) COMP (which translates to 8 bytes on UNIX). However, with TIP/ix, these variables have been all been changed into PIC 9(9) COMP (4 bytes on UNIX).

You must ensure that this translation takes place when transporting the existing applications to TIP/ix. Failure to do so will result in unpredictable results.

### Demand Mode Processing not Supported

The 1100 APIs in this release do not support demand mode processing.

### Compiler Options

To use the 1100/2200 APIs you must link-edit your application with the lib2200.a library. The following makefile is for Micro Focus COBOL:

```
# Modified to handle .tip
#
.SUFFIXES: .c .ims .tip .cbl .cob .bat .rpg .dml .sub

SCHEMA = n0tf0001psch
INC = $(TIPROOT)/include:$GUNHOME/tt/copy
BIN = $(GUNHOME)/tt/live
LIB = $(TIPROOT)/lib/libtip.a
LIBT = $(TIPROOT)/lib/ -l2200
BAT = $(TIPROOT)/lib/libbat.a
```

```
LINK = $(LIB) -lc
LINKT = $(LIBT) -lc -lm
#MFCOB = -gUa -C "IBMCOMP NOWARNING VSC2"
MFCOB = -gx -C "IBMCOMP NOWARNING VSC2"
INCIMS = -I RETURN -I BUILD - I GET -I GETUP
INCTIP = -I TIPPRINT -I TIPFCS -I TIPMSGO -I TIPQUEUE
-I TIPPEER -I TIPRTN

# Following is for IMS transactions using the Micro
Focus COBOL
.ims:
genmain -im $* main$*.c
cob -c $(MFCOB) -k $<
cob $(MFCOB) main$*.c $< -o $* $(LINK)
# rm -f $*.o main$*.o main$*.c

# Following is for TIP transactions using the Micro
Focus COBOL
.cob .cbl .dml:
genmain -m $* main$*.c
cob $(MFCOB) main$*.c $< -o $(BIN)/$* $(LINK)
$(INCTIP)
rm -f $*.o main$*.o main$*.c

# Following is for TIP transactions using the Micro
Focus COBOL
# make the object first, then genmain with -1, then
link
.tip:
cob -c $(MFCOB) -k $<
genmain -1m $* main$*.c
cob $(MFCOB) main$*.c $*.o -o $* -L$(LINKT)
rm -f $*.o main$*.o main$*.c

# Following is for batch Cobol programs
.bat:
cob -c $(MFCOB) -k $<
cob $(MFCOB) $*.o -o $(BIN)/$* $(BAT)
rm -f $*.o
```

## DPS API

TIP/ix supports the following DPS/1100 functions:

**Init & terminate**

D$INIT

D$CLOSE

D$TERM

**Input and Output**

D$OPEN

D$GETFL

D$GETFL1

D$GETLI

D$GETLI1

D$GTSCN

D$READ

D$RETRANSMIT

D$PTSCN

D$RESET

D$SEND

D$SENDF

D$SENDF1

**Paging**

D$CALLPG

D$ENDPG

D$PAGEST

D$RELEASE

D$REPLACE

D$RETR

D$STORE

**Error Handling**

D$DISPLAYERR

D$ENDMSG

D$ERRMSG

D$FLDERR

D$RETURNERR

D$SENDERR

D$USERMSG

Scratch Area

D$GETSCR

D$PGETSCR

D$PPUTSCR

D$PUTSCR

**User Special Field Functions**

D$CLCONV

D$SETCV

D$SETRX

**Destination Selection Functions**

D$PASSOFF

D$ABORT

D$DUMP

**Program Recovery**

D$CLRRCB

D$GETRCB

D$SETRCB

**Integrated Recovery Initialization**

D$INIT2

D$INIT3

**Output**

D$CANCEL

D$CKPT

D$ENQUEUE

D$OUTPUT

D$PASSOFF1

D$PRTLINE

D$SEND1

**Input**

D$DISRCV

D$INPUT

D$RELINP

Recovery

D$COMMIT

D$ROLLBK

**The following DPS APIs are not supported:**

D$INIT1, D$LINIT1, D$FINISH, D$GETIN, D$GETRESET, D$GTEMPLATE, D$GTSCN1, D$PTEMPLATE, D$RESEND, D$SETPAG, D$STOREI, D$USERERR, D$GETDEVICE, D$GETTERM, D$SETDEVICE, D$SETTERM, D$GETUFLD, D$PUTUFLD, D$SETRX1, D$BELL, D$SENDBELL, D$CHGPID, D$CONFIGRET, D$CONFIGSET, D$GETWSDEF, D$RESUME, D$SUSPEND, D$GETTRACE, D$GETWS, D$LOGOFF, D$SCRGET, D$INIT4, D$CLRLST, D$SETLST, D$AUDIT, D$CLLINK, D$SETLINK, D$PIDSTAT.

## Raw Input/Output Functions

The current implementation only supports UTS 400 compatible FCCs in the following raw input functions:

**D$GETFL**
**D$GETFL1**
**D$GETLI**
**D$GETLI1**

The function D$PTSCN does not support the translation of UNISCOPE screen to IBM 3270 or the Dataspeed protocol.

## Paging Functions

The current implementation fully supports OS/1100 paging functionality with one exception: the index pages are not supported. As a result the function D$STOREI is not implemented and in function call D$RETR the negative page numbers (index pages) are not supported.

TIP/ix provides two configuration parameters to enable you to use disk space efficiently:

| Parameter | Description |
|---|---|
| DPSPAGESIZE | Set a limit on the size of a page. An average screen must fit in this space. The default is 4096 bytes. |

| | |
|---|---|
| DPSMAXPAGES | Set the maximum number of pages to be stored in the paging file. The default is 20. |

### Pager Utility

The user interface for the pager is different from the one for the DPS-1100. See tippager in the TIP/ix Utilities manual.

### Scratch Functions

The TIP/ix implementation of the scratch functions (D$PUTSCR, D$GETSCR, D$PPUTSCR, D$PGETSCR) requires the CDA area to be defined. This can be done in the smprog entry of a program using the scratch functions. The size of the CDA defined must be equal to or greater than the size of the internal scratch area used by the program.

## Integrated Recovery Environment (IRE) Functions

To determine if a function is implemented in TIP/ix, see the summary table above. This subsection addresses only the functions that differ from their counterpart in OS/1100.

### Calling Sequence for DPS 1100 Functions

COBOL:

```
CALL 'function' USING   DPS-STATUS,
                        arg2,
                        arg3,
                        arg4
```

**Note:** On TIP/ix the dollar sign ($) in the DPS 1100 function names has been replaced by underscore (_), for example, 'D$INIT' becomes 'D_INIT' on TIP/ix. For consistency, the functions described below still use the original name.

| Function | Difference |
|---|---|
| D$INIT2 | Session control options (RP$OPT2) and parameter RP$PAR3 are not supported. |
| D$INIT3 | Session control options (RP$OPT2) are not supported. RP$PAR4 is set to 1 on return for all cases. |
| D$CLRLST and D$SETLST | Not supported. Output functions such as D$SEND always send a message to the terminal where the DPS application is run. However, the function D$SEND1 accepts user-supplied MDL as a parameter and sends messages accordingly. |
| D$AUDIT | This function outputs text to the TIP/ix log file. |

| | |
|---|---|
| | There is no dedicated audit file for it. |
| D$CANCEL | The whole message will be discarded regardless of the parameter RP$MSGID. The message identifier is not supported. |
| D$CKPT | Messages can be sent in multiple segments. However, segments are sequentially put together to form one single message and no message identifiers are assigned. The parameters RP$AUXLEN, RP$MSGID, RP$MSGNBR, and RP$STEP are not supported. |
| D$ENQUEUE | Messages can be sent in multiple segments. However, segments are sequentially put together to form one single message and no message identifiers are assigned. The parameters RP$MSGID is not supported. |
| D$OUTPUT | Messages can be sent in multiple segments. However, segments are sequentially put together to form one single message and no message identifiers are assigned. The parameters RP$AUXLEN, RP$MSGID, RP$DEV, RP$PDEV, and RP$STEP are not supported. |
| D$PASSOFF1 | Messages can be sent in multiple segments. However, segments are sequentially put together to form one single message and no message identifiers are assigned. The parameters RP$AUXLEN, RP$MSGID, RP$MSGNBR, and RP$STEP are not supported. |
| D$PRTLINE | The parameter destination is not supported. This function always sends output to the TIP/ix default printer PRNTR. The parameters RP$MSGREC, RP$MSGID, RP$PDEV, RP$DEV, and RP$SETP are not supported. |
| D$SEND1 | Multiple destinations are supported. However, if the message is sent to destinations other than the terminal where the application is run, it will be sent as an unsolicited message. Parameters RP$MSGID and RP$STEP are not supported. |
| D$DISRCV | Not supported. A call to this function returns a fatal error. |
| D$COMMIT | The functionality is the same regardless of |

| | |
|---|---|
| | the option specified by RP$OPT1. The parameter RP$OPT2 is not supported. |
| D$ROLLBK | The functionality is the same regardless of the option specified by RP$OPT1 and RP$OPT2. The parameter RP$STEP is not supported. |

### Integrated Recovery Functions

This section describes the differences between OS/1100 integrated recovery and the TIP/ix implementation.

Of all the integrated recovery components, exec audit does not fully function yet and won't be used for recovery. Universal Data System Control (UDS Control) and Integrated Recovery Utility (IRU) are not implemented as components while some of their functionalities such as file restore or recovery are available in TIP/ix.

Currently, the TIP/ix implementation cannot reschedule input messages that the system had not completely processed because of a failure. Nor can it resend output message that terminals did not yet receive. Message recovery is not available.

Application group is not implemented.

Auxiliary data are not supported.

## MCB API

TIP/ix supports the following MCB APIs:

| MCB Function | TIP/ix Function |
|---|---|
| TRINIT$$ | FTRINIT |
| TERM$$ | FTERM |
| RECV$$ | FRECV |
| CANCEL$$ | FCANCEL |
| SEND$$ | FSTORE |
| PASOFF$$ | FPASOFF |
| ENQUE$$ | FENQUE |
| CHKPT$$ | FCHKPT |

The following MCB APIs are not supported:

COMMIT$$, ROLLBAK$$, INPREL$$, AUDIT$$, DISREC$$, OPNSSN$$, CLSSN$$, PIDSTAT$$, SETXUSE$$, CLRXUSE$$, SETIXUSE$$, STAT$.

## Differences in TIP/ix Implementation

MCB programs will require some changes, implemented either manually or through ARM utilities, to be able to run in the TIP/ix environment.

### PACKET copybook

A new copybook called “PACKET” forms the basic block for delivering messages between the MCB program and TIP/ix. Your existing programs may use something similar. However, PACKET will be different in the following respects:

- The names of the fields have been changed from P$FieldName to MCB-FieldName to avoid ‘$’ signs.
- The 9 bit bytes of the 1100/2200 environment have been changed to 8 bit Bytes for the TIP/ix environment.
- For the sake of alignment, some fields were moved around notably MCB-BUFF and MCB-MDL and some filler bytes were added.

The copybook PACKET is available in the global include directory of the TIP/ix environment (/u/tipsrc/include).

### Facility Field (FAC)

The MCB-FAC, the facility field, is always set to 0 indicating normal use of resources.

### Message ID Field

In general, the MCB-API does not support a message recovery mechanism. As a result the values of the MCB-ID, the message id field, will be interpreted as such: 0 indicating a new message while any other positive value indicates a previous message still residing in the buffer.

### Operational Mode

The MCB-MODE, the mode of operation field, supports the normal mode (0) and the test mode (2) only. The training mode (1) and the test/training mode (3) are translated to the test mode (2).

### Transaction Code

The TC1 and TC2 fields have been replaced by an equivalent single MCB-TC field.

### Fields Not Supported

The following fields are not used in the PACKET.

MCB-RTN

MCB-REQUE

MCB-RECOPT

MCB-ERR

MCB-NODE

MCB-RUNID

MCB-OS1

MCB-OS2

## COMPOOL Support

TIP/ix supports the following compool primitives:

| Function | Description |
|---|---|
| CINIT | Initialize the compool environment. |
| CCDTAC | Get the raw message for the application without releasing the message. |
| CCDTCR | Get the raw data and then releases the message. |
| CCSTLG | Store an output message. |
| CCSTOV | Overlays an output message on top of an existing output message. |
| CRELOG | Release any outstanding message. |
| CRTRNO | Send an output message. |
| CTRMN8 | Terminate the transaction. |
| CRTRNU | Pass a message to another transaction. |

The following are not supported:

CRTSCH, CRTOTP.

### Differences in TIP/ix Implementation

#### TIPMPA copybook

The TIPMPA copybook may look slightly different from site to site, so you may need to look at any programs using compool very closely. The TIPMPA copybook accommodates the 9-bit bytes of 1100 systems into the universal 8 bit bytes (with the necessary expansion of fields where required).

## KONS API

TIP/ix enables KONS users to continue using 1100/2200 KONS functions in on-line transactions (under TIP/ix). TIP/ix provides:

- a user KONS area and a security directory.
- the ability for transaction programs to update the KONS security directory and the protected area S2 and U1.

- 16 KONS functions for use with UNIX COBOL programs.

### Parameters

All numeric parameters (COUNT or INDEX) use one-word (four-byte) data items. For example: PIC 9(9) COMP.

### Configuration Parameters

Configuration parameters are environment variables (which can also be set in the TIP/ix configuration file tipix.conf). FALSE is 0 and TRUE is non-zero.

| Parameter | Description |
|---|---|
| KONSBL | Length in bytes of the U3 block in the user KONS area. The default is 0. The value should be an integral divisor of KONU3L to avoid unused space. |
| KONSEC | Length in bytes of the write-protected U1 area of the KONS file. The default is 0. |
| KONSFL | Length in bytes of the U1, U2, U3, and security directory areas of user KONS. The default is 0. |
| KONSPW | Set KONSPW to TRUE to require a password for access to a U3 KONS block. The default is FALSE. |
| KONU3L | Length in bytes of the U3 area of the user KONS file. The default is 0.<br>This area is lockable in blocks of size KONSBL.<br>A password can be required for block access if KONSPW = TRUE.<br>If set to 0, the KONS U3 logic is disabled.<br>If KONU3L is set, then KONSFL must be set to at least KONU3L+4. |
| KSECNB | Number of entries in the security directory for the U3 area of the user KONS file. The default is 0.<br>The security directory is located at the end of U3 area and requires 8 bytes per entry. This space is allocated regardless of the value of KONSPW. |

### KONS File Layout

The layout of KONS file is shown below, where:

```
SECURL = KSECNB*2.
```

### Structure of Directories

The structure of the KONS directory is shown below:

These functions are added for handling DIRECTORIES. The functions can be called using ‘CALL “CKONS”’ from COBOL programs. These

functions are intended for internal use for writing a utility that handles the KONS file (initialize, test, load, and update).

DIRRD DIRECTORIES READ. It has the following one parameter:
BUFFER Area that holds the entire DIRECTORIES table.
DIRWR DIRECTORIES WRITE. It has the following one parameter:
BUFFER Area that holds the entire DIRECTORIES table.

Another version of the format is that the end block number is stored instead of the number of blocks. However, the above is the version that is used in TIP/ix KONS.

### KONS Functions

There are 16 KONS functions for accessing the KONS file. For details, see the OS 1100 Transaction Processing Programming Reference manual (UP-8296.8-C).

| KONS Function | Notes |
|---|---|
| CUPDAT | |
| FILLWR | |
| KRDLK | |
| KREAD | |
| KWRITE | |
| KWRKP | |
| KWRUN | |
| LUPDAT | |
| MSREAD | |
| SCATRD | |
| SEQSRD | |
| SERCHR | |
| SREAD | |
| SUPDAT | |
| SWRITE | |
| UNLOCK | UNLOCK is implemented as UNLOCKX (because UNLOCK is a reserved word in COBOL). |

### copybook for COBOL Programs

There is a copybook for COBOL programs which use TIP/ix KONS functions. This copybook, SYSDEF, defines function codes including

CUPDAT, FILLWR, KRDLK, KREAD, KWRITE, KWRKP, KWRUN, LUPDAT, MSREAD, SCATRD, SEQSRD, SERCHR, SREAD, SUPDAT, SWRITE, and UNLOCK.

### S and S + Q Options

The S Option is implemented as the security level SYST and the S + Q Option is implemented as the security level MAST. The security check follows the usual rules used in TIP/ix. One significant difference is that the user's security overwrites that of the transaction. For example, if the transaction using CKONS functions is at the security level PROG while the user is at SYST, when the user runs the transaction, the process gets security level SYST which means the S Option is set.

### Error Messages

There are 15 system error codes for KONS in OS 1100. Those that are implemented in TIP/ix are indicated by *.

| Code | Description |
|---|---|
| 01 | Program attempted an operation outside the KONS area. |
| 01 | Program attempted an operation outside the KONS area. |
| 02 | Program attempted to write into S1 portion of system KONS. |
| 03 | Program attempted to write into S2 portion of system KONS or U1 area of user KONS without "write" option set. |
| 04 | Program's buffer is outside the program's limits. Specified buffer length exceeds the amount actually provided. |
| 05 | Illegal partial-word designator or partial-word descriptor. |
| 06 | Illegal function code. |
| 07 | Program attempted to use a search function in the system KONS area. |
| 10 | Illegal security operation. The option bit not set for writing or reading into the security directory, or attempt has been made to perform any security operation without proper security code. |
| 11 | Attempt to access U3 area with incorrect function. |
| 12 | Attempt to perform any lock function past the end of the block. |
| 13 | Requested block is not locked. |
| 14 | Attempt to allocate request buffer failed, or reject was received while attempting to read the file directory. |

15 File is locked.

16 User has maximum number blocks allowed.

17 Excessive activity caused maximum number of retries to obtain lock.

## FCSS API

This section includes the on-line FCSS functions. These functions are also available to batch connected programs.

**UCS COBOL**

```
CALL "FCSS" USING       func,
                        return,
                        buffer,
                        fileNumber,
                        wordCount,
                        bufferSize
```

**ASCII COBOL**

```
CALL "CFCSS" USING      func,
                        return,
                        buffer,
                        fileNumber,
                        wordCount,
                        bufferSize

    or

CALL "CBFCSS" USING     func,
                        return,
                        buffer,
                        fileNumber,
                        wordCount,
                        bufferSize
```

**Where:**

**func** Function code

**return** Logical file name packet.

**buffer** Record area where status information and line data is placed

**fileNumber**
Number of file to access

**wordCount**
Count of data words in buffer

**bufferSize**
Count of total words in buffer; ignored

The function code values and return code values are defined in the TIP/ix copybook TC-FCSS.

The bufferSize parameter is no longer used in 1100/2200 environment and is not validated.

The buffer argument in 1100/2200 contained 3 words of status and file information followed by a user determined amount of data, the wordCount parameter specifying how much data was available. Unavoidably the format of the buffer status area has changed and is in fact larger in the TIP/ix environment. Careful attention must be paid to the buffer argument in the conversion process. There was no 1100/2200 standard copybook describing this region, but one has been added for TIP/ix, it is TC-FCSSU.

```
*****************************************
*                                       *
* TIP/ix FCSS User Buffer copybook      *
*                                       *
*****************************************
02  FCSS-BUFFER-STATUS.
    10  FCSS-BLOCK-ZERO.
        15  FCSS-ERROR-FLAG     PIC 9.
            88  FCSS-GOOD           VALUE 0.
            88  FCSS-ERROR          VALUE 1.
        15  FCSS-RECYCLE-FLAG   PIC X.
        15  FILLER              PIC X(9).
        15 FCSS-COMPLETE-FLAG   PIC 9.
            88  FCSS-NOT-DONE       VALUE 0.
            88  FCSS-DONE           VALUE 1.
        15  FCSS-RQSTAT         PIC 9(4) COMP.
        15  FCSS-PFD.
            20  FCSS-BIT-17     PIC X.
            20  FCSS-BIT-16     PIC X.
            20  FCSS-BIT-15     PIC X.
            20  FCSS-BIT-14     PIC X.
            20  FCSS-BIT-13     PIC X.
            20  FCSS-BIT-12     PIC X.
        15  FCSS-FCSSD          PIC 9(4) COMP.
        15  FILLER              PIC X(2).
    10  FCSS-BLOCK-ONE.
        15  FCSS-L2STAT         PIC 9(4) COMP.
        15  FCSS-L1STAT         PIC 9(4) COMP.
        15  FCSS-DIADR          PIC 9(8) COMP.
    10  FCSS-BLOCK-TWO.
        15  FCSS-BTMASK         PIC 9(8) COMP.
02  FCSS-BUFFER-DATA.
```

### Supported FCSS functions

| Func | Description | Notes |
|---|---|---|
| AD | Write a user record to the journal file. | |
| FL | Lock a permanent file from all accesses by other programs. Your program accesses the file through the RD and WW functions. | |
| FR | Lock a permanent file from all but RD accesses by other programs. | |
| LK | Create a data lock on the referenced records. | There was a system parameter FCMXLK which limited the maximum number of locks per user, this is not currently supported. |
| RD | Read one or more records. No record locking is performed or checked on the referenced records. | |
| RE | Release a permanent, temporary or scratch file. | Permanent file maintenance will be done via the smfile utility only, so the release function will be limited to the temporary and scratch FCSS files. |
| RL | Create a data lock on the referenced records and read the program buffer. | |
| RR | For programs using ASCII COBOL, use RR instead of RD. | |
| UN | Unlock a record lock or an FCSS file lock or all locks held by your program. This function currently works for non-distributed transactions. | |
| WL | Write a record and keep the record or file lock already held by your program. | |
| WR | Write a record or records. | |

| | | |
|---|---|---|
| | The records must have been previously locked by your program. | |
| WW | Write a record without checking for a record lock. | This operation is invalid for permanent fixed FCSS files. |

### Unsupported FCSS functions

| Func | Description | Notes |
|---|---|---|
| AL | Allocate a Freespace record. The record is locked before control is returned to your program. | |
| AN | Allocate a Freespace record, lock it, write data to it, and then release the lock. | |
| AP | Allocate a specific Freespace record by its record key. | |
| AS | Assign a temporary or scratch FCSS fixed file in an Exec file. | |
| AW | Allocate a Freespace record, lock it, and write data to it without releasing the lock. | |
| CK | Check for the completion of one or more requests your program issued with an immediate return type. | This function only has meaning if various return types are supported, therefore it's not planned. |
| LF | List the characteristics of a permanent, temporary, or scratch file. | Accessible with smfile print option, unless required within a transaction. |
| RU | Unlock a Freespace record and return it to the system as a free record. | |
| SF | Save on disk any TIP file updates. | Similar to the FCS-FLUSH function which was not ported to TIP/ix because UNIX handles file flushing. |
| UA | Release all record and file | |

locks at commit time.

## Differences and Implementation Details in TIP/ix

### File Types

Fixed FCSS files can classified as permanent, temporary or scratch. The permanent fixed FCCS format files are supported. They are defined with smfile as FCS Direct files.

Temporary and scratch FCSS files are not supported.

Freespace files are not supported.

TCDBF (TIP common data bank) files are not supported.

### Return Codes

OS 1100 FCSS function calls allow for specifying when control should return to the application;

| Code: | Description |
|---|---|
| FCDONE: | Control returns after the completion of the request this represents. |
| FCIMED: | Control returns immediately, the program must check for the completion of the request. |
| FCALLD: | Control returns after the completion of all outstanding FCSS requests |
| FCDNIL: | Synchronous read of record after immediate locking. |
| FCIMIL: | Asynchronous read of record after immediate locking, |
| FCALIL: | Completes all outstanding FCSS requests after immediate locking. |

The only return code supported will be FCDONE. Function calls are validated based on the old return code requirements. The last three modes are used on a RL (Read and Lock function) to return immediately, otherwise the RL will wait forever for the lock.

### File Naming

The OS 1100 file system refers to all files by a TIP file number. The file numbers range in value from 17 to a system defined maximum, the first 16 numbers are reserved for system use.

To relate the file number to a TIP/ix file name, the name as entered with smfile must contain a 0-filled numeric value “00000100” and the path field will contain the actual file UNIX file name, perhaps “/u/tipsrc/tipfiles/fcss100”. The valid file numbers will range from 1 to 99999999. The maximum value is derived by the maximum number of characters in the file name field.

### Maximum file size

Files are pre-allocated and the size can not be expanded due to record writes. When a file is registered with a specific number of records and a specific record length, its size has been established. If an attempt is made to write beyond this size, an error is returned.

To enforce a maximum file size, you can use smfile to specify the maximum number of records. If the maximum record count has been added to the file definition it will be used to limit writes to the file. The records are pre-allocated and filled with dummy values. If the file already exists and exceeds the specified maximum a warning message will be logged and a maximum record count true to the file's actual size will be used.

### Exclusive File Access

OS 1100 files are only accessible through the FCSS interface and through batch programs which have connected to the TIP system. Files should be defined with Exclusive access via smfile.

### Record Locking

The record unlock function UN unlocks a record lock or an FCSS file lock or all locks held by your program.

However, if there have been any updates against the record locks to be released, the request is ignored. This function currently works for non-distributed transactions. If it is used in a distributed environment results are unpredictable.

### Deadlock Handling

In TIP/ix, when deadlock is detected an error is returned but no current locks are released.

In OS 1100, when a potential deadlock is detected on a record lock function, an error is returned, and the other lock which contributed to the deadlock situation is released as well. The releasing of the lock is not supported because this interferes with TIP/ix system design.

### Unprotected Files

The WW function writes a record to an unprotected file regardless of locks held by this or another program. An unprotected file is one contained in a TIP file table that has the "WW allowed" option set..

This function is invalid for permanent fixed FCSS files and Freespace files, because the file may be used in a recoverable transaction and this kind of operation would corrupt the data.

### Duplex files

Files can be either simplex or duplex, this is a file mirroring technique. The duplex file type is not supported.

### Maintenance functions

Specific maintenance functions allow the transaction to modify pre-registered file attributes. The RV reserve a permanent file, CG change a permanent file and RE release a permanent file are back door functions to modify the OS 1100 VALTAB entries for a file. In TIP/ix, you use smfile to achieve the equivalent operation.

The LF list characteristics function is not supported. Use the Print option in the smfile menu instead.

### Error handling

Many error codes available in OS/1100 have no meaning in the TIP/ix environment. The error status bit will indicate success or failure of an operation correctly, however, various other codes may not occur as expected. You should review any specific error code handling.

## Transaction Processing API

**Commit:**

```
CALL "COMMIT"
```

or the ASCII COBOL equivalent

```
CALL "CCMMIT"
```

**Rollback:**

```
CALL "ROLLBACK"
```

or the ASCII COBOL equivalent

```
CALL "CRLBACK"
```

**Where:**

| Function | Description | Notes |
|---|---|---|
| COMMIT | Commit updates, all locks held by this transaction are freed. | This is equivalent to the TIP/ix FCS function FCS-TREN with the user's PIB lock indicator set to PIB-COMMENT |
| ROLLBACK | Rollback updates, all locks held by this transaction are freed.. | This is equivalent to the TIP/ix FCS function FCS-TREN with the user's PIB lock indicator set to PIB-ROLLBACK |
| CCMMIT | ASCII COBOL equivalent to COMMIT. | |
| CRLBACK | ASCII COBOL equivalent to | |

ROLLBACK

### Batch Connected FCSS API

In batch connected mode the FCSS API is the same, the following functions are provided to allow the user to connect and disconnect to TIP/ix. To use the batch connect mode you must link your applications with the libbat.a library.

**Connect:**

```
CALL "CONECT"
```

or the ASCII COBOL equivalent

```
CALL "CCONET"
```

**Disconnect:**

```
CALL "DISCON"
```

or the ASCII COBOL equivalent

```
CALL "CDISCN"
```

**Where:**

| Function | Description | Notes |
|---|---|---|
| CONECT | Connect program to TIP/ix. | This is equivalent to the TIP/ix BATFCS function BATACTIV or BATCONNECT. |
| CCONET | Equivalent to CONECT. | |
| DISCON | Disconnect program from TIP/ix | |
| CDISCN | Equivalent to DISCON. | |

## Miscellaneous

### STATUS & TIPDUMP Utilities

The status and tipdump utilities have been modified to provide some extra information for 1100/2200 applications. Both of these utilities now provide sub-transaction Id and the type of the lock (exclusive or read-only) being applied to the file. To get this information, execute status -k.