Suprtool 6.2 for MPE:

User Manual

by Robelle Solutions Technology Inc.

Robelle Solutions Technology Inc.
Suite 372, 13711 – 72nd Avenue,
Surrey, B.C. Canada V3W 2P2

Phone: 604.501.2001
Support: 289.480.1060

E-mail: sales@robelle.com
E-mail: support@robelle.com
Web: www.robelle.com

Table of Contents

Dbedit Commands 413

Welcome to Suprlink 438

Accessing Suprlink 440

Welcome to Suprtool

Introduction

Welcome to version 6.2 of Suprtool — the HP e3000 handyman for Image and Allbase databases, MPE files, and KSAM files. Suprtool provides fast batch reporting on your HP e3000. With Suprtool, you can select records from a file (IMAGE, Allbase, KSAM, or MPE) to feed into your report program. Typically, Suprtool enables you to produce the same report five times faster.

Suprtool performs best when you are selecting less than 50% of the entries. Once selected, you can rearrange, sort, print, or total them. You need to modify your report program to read a file instead of a dataset.

In addition, Suprtool solves dozens of other problems. The Dbedit subsystem allows convenient, interactive editing of databases, the Suprlink subsystem allows multiple dataset access, and the Speed Demon intrinsics allow a user program to improve the speed of serial DBGET calls by a factor of five. Use STExport to convert fields from a self-describing input file to an output file that can be imported into other applications. The interface routine Suprtool2 allows your application programs to invoke Suprtool.

The Suprtool commands are:

Add	EDit	ITem	Q	Use
BAse	Exit	Key	REDO	USERpause
Before	EXPort	LINk	Reset	Verify
Chain	EXTract	List	SELect	Xeq
Clean	Form	LISTREDO	Set	:*MPE or HP-UX command*
Define	Get	Numrecs	SOrt	*=expression*
DELete	Help	OPen	TAble	
DO	IF	Output	Total	
Duplicate	Input	Put	UPdate	

The minimum abbreviation of each command is shown in capital letters.

Suprtool Components

The Suprtool package consists not only of Suprtool, but also of other programs that perform useful database functions. These other programs are Dbedit, Speed Demon, STExport, Suprlink, and Suprtool2.

Database Editing

Dbedit is a separate module of Suprtool that permits the user to add, change, list, or delete individual records or "chains" of records from an IMAGE database. You enter

Dbedit via the Edit command of Suprtool, after specifying a database with the Base command. The Dbedit prompt character is #, not >, and the commands are:

Add	File	Q	:*mpe command*
Before	FOrm	REDO	=*expression*
Change	Help	Set	
Delete	List	Use	
DO	LISTREDO	Verify	
Exit	Modify		

Speed Demon - Replacing DBGET Mode-2

One part of the Suprtool package is Speed Demon, a set of intrinsics that you call in your programs instead of DBGET mode-2 to read serially through a dataset; it should be about five times faster.

While Speed Demon lacks Suprtool's ability to extract fields or select and sort records, it has the advantage that it delivers the records directly into your program. Speed Demon/V is about 50% slower than Suprtool, but there is no need to read an output file, as with Suprtool. Speed Demon/iX runs at the same speed as Suprtool. With Speed Demon, your application program is responsible for selecting records and sorting them (e.g., using the Sort verb in COBOL).

Application programs should invoke Suprtool when they expect to select a **small percentage** of the records (i.e., when the overhead of reading the output file will be minimized). Application programs should call Speed Demon when they expect to select a **large percentage** of the records.

STExport - Data Export Utility

STExport converts fields in a self-describing input file into an output file that can be imported into different applications.

Use STExport to produce a formatted output file that can be used to import data into databases and applications.

Other databases have different requirements for the format of input data. You will have to experiment with the various STExport formatting options to find a format that your particular database tool accepts.

Rport - Data Reporting Utility

Rport converts fields in a self-describing input file into an output file that is automatically aligned based on the size of the fields.

You can use Rport to produce simple reports with rules for formatting most data fields as well as subtotals and totals for any numeric field. Dates can be formatted and converted to most formats on the fly.

Titles and Headings are also available with a minimum of commands and formatting.

Suprlink - Multidataset Access

Suprlink is a program that works with Suprtool to add "multidataset" capability to Suprtool. Suprlink is not a set of callable routines. To use it, you must :run Suprlink.Pub.Robelle or use Suprtool's Link command.

Rather than take the regular path to multiple datasets -- random retrieval via IMAGE keys -- with its well-known performance problems, we have chosen to follow a different path: fast serial extracts plus a very efficient merge. The tests that we have performed indicate that this method is often significantly faster than the "official" IMAGE method of chasing down chains and hash synonyms.

To understand what Suprlink does, think of the process of writing a report. Your report program (written in COBOL, RPG, PowerHouse, or some other language) hunts all over the database with DBFIND and DBGET to collect your data.

It would be faster if the report program could just read a sorted disc file with a big record containing all the data necessary for the report, and this is Suprlink's function. Suprtool can extract the desired fields from the desired records of the sales detail dataset and put them in a disc file. Then Suprtool can extract the desired fields from the customer master dataset and write them to a second disc file. If Suprtool sorts both files by customer, Suprlink can "link" them together, producing a third file whose composite record consists of the related fields from both files. This file is just what we need to feed into the report program. For example, a sales report program might read a disc file whose records consist of sales transactions plus customer information. This file has been sorted by customer number and date. If there are several sales for the same customer, the customer information is just repeated in each record. The report program reads the records, checks for level breaks, and formats and prints the records.

Suprtool2 - Interface Routine

How would a user application program invoke Suprtool to perform a desired task? Robelle provides an interface routine that runs Suprtool for a user program, passing commands from the program to Suprtool (the same commands you would type into Suprtool). This routine (or procedure, subroutine, intrinsic) allows user programs to call Suprtool. A typical use of this interface would be for a COBOL program to ask Suprtool to extract a selected subset from a large IMAGE dataset and write it to a disc file, which the COBOL program would then read and format into a report. User programs written in COBOL, TRANSACT, Speedware, SPL, FORTRAN, Pascal, COBOL/iX, FORTRAN/iX, or Pascal/iX can call Suprtool2.

Documentation

The user manual contains the full description of all the Suprtool suite of products including Suprlink, STExport, Dbedit, Rport and Suprtool2, as well as usage tips and commands for each. The manuals are up to date with all the latest changes. To see only the changes in the latest version, see the "What's New" section of the manual, or see the change notice.

You can download our manuals in PDF format or HTML format and even order printed (hardcopy) manuals from our web site at:

http://www.robelle.com/library/manuals/.

Notation

The Suprtool documentation uses a common notation in describing all commands. Here is a sample command definition:

EXTRACT *field* [(*subscript*)] [=*value*] [,...]

1. UPPERcase letters - literal symbols to be used in the command as they appear (e.g., EXTRACT).
2. Lowercase, underlined or italic - "variables" to be filled in by the user (example: *field*). Each such "variable" is defined elsewhere in terms of literal symbols (consult the index). In the help file, underlining and italics are not available and variables appear simply in lowercase.
3. Brackets - enclose optional fields (example: [(*subscript*)]).
4. Braces - enclose comments in examples. For example, >INPUT ACTREC {input from a data file}. Braces **can** be used for comments in actual Suprtool commands.
5. Up lines - separate alternatives from which you select (example: Set Ignore [On|Off]); sometimes, the alternatives are shown listed on several lines.
6. Dot-dot-dot (...) - indicates that the variable may be repeated many times in the command.
7. Other special characters - literal symbols that must appear in the command as they are shown in the format (example: the comma above). Some commas in Suprtool are optional.

In examples, there is an implied carriage return at the end of each line.

Examples

The examples in this manual use the revised STORE database described in the **IMAGE/3000 Handbook**.

Suprtool Version 6.2

Highlights in Suprtool 6.2

- Rport has a new command called Hide which is intended to not report on a field in an SD file.
- Rport has a new command called Comma, which will place appropriate commas in an ascii field that was converted from a numeric field.
- There is a new setting in RPORT that is called set maxlen on, which calculates the maximum size that a field can be, including commas and dollar signs.
- Suprtool/Open now defaults to /bin/bash when a users shell is not found in the user information.
- The $eom/$bom function wouldn't work properly when $stddate was nested in a $month function and the to month had 31 days.
- Sdlinux version is available for Linux to convert the .sd file for use on Linux is now available on Linux. You can now directly copy a data file and sdfile from HP-UX and convert it for use on Linux.
- Two new sub-functions of $BOM and $EOM are now available in the $stddate and $month functions. The $BOM function returns the starting day of the month and the $EOM returns the last date of the month for a given month.
- String functions in Suprtool now have better length and bounds checking of the 4095-byte string limit.
- Rport a new simple report writer has been added to the Suprtool suite of products.
- Set Limits Override has been added to the Suprtool for MPE version. (6.1.01)

Highlights in Suprtool 6.1

- Suprtool has a new function called $length that will return a double integer value of the length of a byte and/or numeric field.
- Suprtool has a new option called Set EditSignNeutral which tells the $edit function to treat Neutral zoned and packed fields to be positive and treated as such by the $edit function.
- Suprlink now supports 8192 bytes on both the input file and the link file.
- Suprlink now supports 16384 bytes on the output file in Suprlink
- Stexport now supports 8192 bytes on both the input file and output files.

Highlights in Suprtool 6.0

- STExport has a new set command called Set Excel Leadzero On which tells the Excel command to add leading zeroes to the fields specified in the Excel Preserve command.
- Suprtool has a new function available to the if/extract commands called $inrecnum, which expects a double integer result.
- Suprtool has a new function called $leadzeroz, which will add leading zeroes to a display field and will optionally justify the field.
- Suprtool has a four new string handling functions, specifically, $justifyl, $justifyr, $leadzerob and $respace.

Installing Suprtool

Overview

Installation of all Robelle software is done vial the web. The installation instructions for each product and download is directly under the download page.

Instructions

The Suprtool production release can be downloaded from our web site via the What's New Page and Request for Codes form.

http://www.robelle.com/products/whatsnew.html#Suprtool

The instructions can be found here, for the two download/upload methods. For the ftp download method:

http://www.robelle.com/downloads/install-stprod-ftp.html

and here for the WRQ download method:

http://www.robelle.com/downloads/install-stprod-wrq.html

Getting a Quick Start with Suprtool

How to Run Suprtool

Use the following command to access Suprtool:

```
:run suprtool.pub.robelle

SUPRTOOL/Copyright Robelle Solutions Technology Inc. 1981-2001.
(Version 6.1)  SUN, JUN 17, 2007, 11:47 AM  Type H for help.
Today's Hint. To see ALL of the options available in Suprtool,
use >VERIFY ALL.
>
```

Suprtool prints its version number and the current time right after a banner Suprtool then prompts with ">". Press Return after typing each command. For example, if you type the help command:

```
>help
```

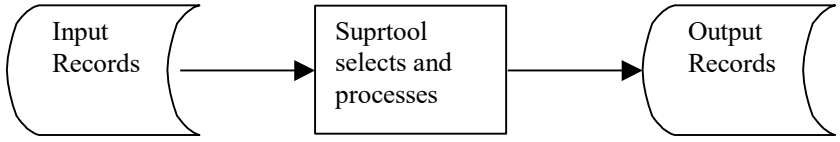
Suprtool prints some help text and a keyword list. Type a keyword or press Return to leave Help.

To exit Suprtool, type "Exit" at the Suprtool prompt.

```
>exit
```

What is a Task?

Tasks are the building blocks with which Suprtool helps you to solve data processing problems. In a task, Suprtool reads information from a file or database, selects and processes some information, and writes out the result. You can visualize a Suprtool task like this:



The examples that follow all consist of Suprtool tasks. Simple solutions require only one task. Complex solutions consist of several tasks, often with the output of one task becoming the input for the next task.

Copying Files

Copying One File

Use the Input command to specify a data file.

```
>input  file1
>output result
>xeq
```

The Output command creates the file called "result", which is a copy of the input file.

Appending to a File

To append to an existing file, use the Append option in the Output command.

```
>input  file2
>output result,append
>xeq
```

Concatenating Two Files

To concatenate two files, you have to tell Suprtool two things:

1. The size of the resulting file.
2. Not to truncate the empty space at the end of the resulting file.

Once you've done this, concatenating two files is just like copying a file, then appending to a file.

```
>:file  result; disc=50000              {maximum 50,000 records}
>set    squeeze off                     {don't truncate space}

>input  file1                           {copy the first file}
>output result
>xeq

>input  file2                           {append the second file}
>output result,append
>xeq
```

Fields in Data Files

What is a Self-Describing File?

A self-describing (SD) file is a file that contains field information. These files have the advantage of behaving like data files, which can provide field information to Suprtool without you having to Define all the fields.

Creating an SD File

To create an SD file, use the Link option in the Output command.

```
>get    d-sales
>output result,link
>xeq
```

Now the data file "result" has the same field names as the dataset d-sales. Suprtool can read this data file and know about the fields automatically.

```
>input  result
>if     sales-total>20000 and product-price<5000
>output custlist
>xeq
```

Define Fields in a Data File

A regular data file does not have any field information associated with it. If you need to work with the fields in a data file, you need to tell Suprtool about the fields using the Define command. For example, say you have a data file with lines that look like this:

```
12345678John         Rutherford     <32>
98765432Catherine    Smith          <29>
|        |           |               |
Account  First name  Last name       Age
8-byte   12-byte     16-byte         2-byte integer
```

Use these Define commands to tell Suprtool about the fields:

```
>input  datafile
>define account,    1,  8, byte
>define first-name, 9, 12, byte
>define last-name, 21, 16, byte
>define age,       37,  2, int
         |         |    |    |
     field name    |  Length |
                 Start       Data-type
                 position
```

Now you can use the field names "account", "first-name", "last-name", and "age" to refer to the corresponding parts of the line, just as if this were a database record.

```
>input  datafile
>define ...
>if     age>65
>ext    account, last-name
>output result
>xeq
```

Create an SD File from a Data File

To create an SD file from a data file, follow these steps:

1. Define the fields that you want to include in the SD file.
2. Extract the fields in the order you want.
3. Use the **Link** option in the Output command to create the SD file.

```
>input   datafile
>define  account,    1,  1, byte
>define  first-name, 9, 12, byte
>define  last-name, 21, 16, byte
>define  age,       37,  2, int

>extract account, age, first-name, last-name

>output  result,link
>xeq
```

Repeating Commands

Repeating a Command

Use the Listredo command to see a list of your most recent commands. Use the Do command to repeat a command or use the Redo command to modify a command before repeating it.

```
>listredo                                  {20 most recent commands}
>listredo input                            {most recent Input commands}

>input result
>...
>xeq
>do  input                                 {repeat previous Input command}

>input result
>if quantity > 10000
>...
>redo if                                   {modify previous If command, then repeat}
```

If you have used two commands that begin with the same letter, you have to be careful when repeating those commands. Make sure you use enough letters to identify each command distinctly from the other. In the following example, if you wish to repeat the Input command instead of the If command, you need to use "do i s" instead of just "do i".

```
>i somefile
>if <expression>
>...
>xeq
>do i s                                    {repeats previous Input command}
```

How to Save On-line Commands to a File

After you have performed a task on-line, you can save the commands you've used to a file. You can edit this file to help you create Suprtool batch jobs.

```
>listredo all;unn;out=somefile
```

Selecting Database Records

These examples show you how to get records from an IMAGE dataset. It assumes you have opened the demo database with the Base command. The results are written to a data file called "result", which can be read either by a program or by a report writer.

```
:run suprtool.pub.robelle
>base store.demo.robelle , 5 , READER  {read access}
```

Select All Records

This example extracts all the records from the dataset. Note that we didn't specify any selection criteria, so Suprtool selects all the records.

```
>get    d-sales                          {input dataset}
>output result                           {output file}
>xeq                                     {Xeq command performs the task}
```

Select a Random Sample

Sometimes you may need to select a random sample from your dataset to test a new report or program. You can easily do this by adding a *#count* parameter to the Get command. The *#count* value tells Suprtool to select only every nth record. The example shows you how to select 5% of your database.

```
>get    d-sales(#20)                     {every 20th record is 5%}
>output result
>xeq
```

Look at the First Few Records

If you want to look at the first few records of a dataset, use the Numrecs command. This command tells Suprtool to extract at most the number of records specified. Then, instead of sending the result to a file, send it to the screen with "output *,ascii". The example shows you how to look at the first 10 records in your dataset.

```
>get     d-sales
>numrecs 10                              {first 10 records}
>output  *,ascii                         {output to screen, format numbers}
>xeq
```

Selecting by Criteria

Simple Criteria

To tell Suprtool to choose records based on certain criteria, use the If command. Think of the If command as a shorthand for "Select all records that have...". In this example Suprtool extracts all records with a sales-total value greater than 20000 from the d-sales dataset.

```
>get    d-sales
>if     sales-total > 20000  {select}
>output result
>xeq
```

Complex Criteria

To choose records using more complex criteria, combine several simple criteria using AND, OR, NOT, and parentheses. In this example Suprtool extracts all records that have a sales-total value greater than 20000 and a product-price value less than 5000 from the d-sales dataset.

```
>get    d-sales
>if     sales-total>20000 and product-price<5000
>output result
>xeq
```

String of Digits

If you have a byte-type field that consists entirely of digits, and you want to use this field as a numeric field, you need to define a new display field on top of the existing field. For example, suppose your data looks like the following, where the customer account number is stored in the 8-digit byte-type field at the start of the record:

```
20476789...rest of customer record...
```

To find all customers with an account greater than 20470000, you would do the following:

```
>get    dataset
>define accountnum, account, 8, display
>if     accountnum > 20470000
>output result
>xeq
```

Selecting by Date

Because IMAGE doesn't provide a built-in date type, most databases store dates as numeric or character fields.

Before Suprtool can use a date field, it has to know the format of a particular date field. Use the **Item** command to specify the format. For example, to tell Suprtool that the item purch-date is a date field with a format of yyyymmdd (e.g., 20010319), you would use:

```
>item   purch-date, date, yyyymmdd      {date format}
```

In the following date examples, we show the Item command in each example. In practice, however, you only need to use the Item command once per date field, not once per task.

Select by Today's Date

For this example, select all the sales records whose purchase date is today. Note the use of $today in the If command to indicate today's date.

```
>get    d-sales
>item   purch-date,date,yyyymmdd
>if     purch-date = $today             {select today's date}
>output result
>xeq
```

Other tricks with $today

```
>if     purch-date = $today(-1)         {yesterday}
>if     purch-date = $today(+1)         {tomorrow}
```

Select by Particular Date

To specify a particular date, use the $date function in the If command. The $date function has the form $date(*year /month /day*). This example selects all the sales transactions for August 12, 2000.

```
>get    d-sales
>item   purch-date,date,yyyymmdd
>if     purch-date = $date(2000/08/12)
>output result
>xeq
```

Select by Year

Suppose we want to select all the sales transactions for 2000. Suprtool does not have a shorthand for specifying "everything in that year". To specify an entire year, use a date range from January 1st to December 31st.

```
>get    d-sales
>item   purch-date,date,yyyymmdd
>if     purch-date >= $date(2000/01/01) and &
        purch-date <= $date(2000/12/31)
>output result
>xeq
```

Select Prior Month

In the $date function, use a * to indicate the current year, month, or day. Similarly, a *-1 means the previous year, month, or day. For this example, select all the sales transactions for the prior month. Note the use of the special keywords "first" and "last" to indicate the first and last day of the month.

```
>get    d-sales
>item   purch-date,date,yyyymmdd
>if     purch-date >= $date(*/*-1/first) and &
        purch-date <= $date(*/*-1/last)
>output result
>xeq
```

Selecting by Lists of Values

Sometimes you want to find records based on criteria contained in another file or dataset.

Finding Data Based on a List

Suppose we want to find all orders for the customers "1234", "9876", and "5555." We simply use a list of values after the equal sign in the If command. A match is made if a customer matches any one of the values in the list.

```
>get    ord-details
>if     cust-no = "1234", "9876", "5555"
>output orders
>xeq
```

If we wanted to find orders for all customers except "1234", "9876", and "5555", we would simply change the equal sign in the If command to a not-equal sign. A match is made if a customer does not match any values in the list.

```
>get    ord-details
>if     cust-no <> "1234", "9876", "5555"
>output orders
>xeq
```

Finding Data Based on a File

If you have a large list of values in a file, you can load them into Suprtool and select data based on this list. First use the Table command to load values from an external file into a table. Then use the function of the If command to match data to the table.

Suppose our list is in a file called Custlist. The table we create is called cust-table.

```
>get    ord-details
>table  cust-table, cust-no, file, custlist
>if     $lookup(cust-table, cust-no)
>output orders
>xeq
```

If you want to find all customers not on the list, just negate the If condition.

```
>if     not $lookup(cust-table, cust-no)
```

You can also compare both a key value and data using the Table data feature.

```
>get    ord-details
>table  cust-table, cust-no, file, custlist,data(state-code)
>if     $lookup(cust-table, cust-no, state-code) = state-code
>output orders
>xeq
```

What the above code does is load a table with key values and data, then as you are reading serially thru the database, Suprtool looks up the key value in the table and retrieves the associated data and returns it and compares against the field on the right-hand side of the equation. If no entry is found in the table then Suprtool returns spaces for byte type fields and zero for numeric fields. This means of course that in our example that if a cust-no did not exist in the table, the state-code returned would be blank and if the state-code was blank in the ord-details dataset the record would qualify.

If you don’t want to have any values returned from the table lookup, you just preface the if with a standard lookup.

```
>get    ord-details
>table  cust-table, cust-no, file, custlist,data(state-code)
>if     $lookup(cust-table,cust-no) and &
       $lookup(cust-table, cust-no, state-code) = state-code
>output orders
>xeq
```

When using $lookup to return data, the $lookup must always be on the right side of the equation. If not Suprtool will stop with an error:

```
If id-field=$lookup(mytable,char-field,id-field)
Error:  $lookup in this context (data comparison) must be on left side
```

Finding Data Based on Another Dataset's Criteria

Sometimes you need to find data from one dataset based on conditions from another dataset. This is a typical example: you want to find all of the pending orders for those customers whose accounts receivable balance is 0.

First we find the customers with an AR balance of 0 and extract their customer numbers to a file.

```
>get    cust-ar
>if     ar-balance = 0
>ext    cust-no
>output custlist
>xeq
```

Now we can find information by loading a file of customer numbers into a table and then applying the $lookup feature.

```
>get    ord-details
>table  cust-table, cust-no, file, custlist
>if     status="PENDING" and $lookup(cust-table,cust-no)
>output orders
>xeq
```

Finding Data in a Data File

So far, the examples have looked up data from a database. If you want to look up information in a data file, you need to tell Suprtool about the fields. Use the Define command to do this.

The following example gives you some idea of the byte-size of one kind of record in a data file.

```
John         Smith           12345678
Anna-May     Richardson      98765432
12-bytes     16-bytes        8-bytes
```

If you want to look up customers based on a list of customer numbers in the file Custlist, use the following task. Notice how the start position and number of bytes is entered into the Define command. This defines the position within the input file, not the table file.

```
>input  flatfile
>define cust-no, 29, 8, byte
>table  cust-table, cust-no, file, custlist
>if     $lookup(cust-table, cust-no)
>output result
```

Comparing against a Key and Data

Suprtools can use the “data” loaded into a Table in a comparison operation. The $lookup function will return the data value from the table to compare against another field or literal.

```
In file1sd
Table mytable,char-field,data,tabfile,data(id-field)
If $lookup(mytable,char-field,id-field) = int-field
```

So, what Suprtool will do in this case is read a record, lookup the record in the table and retrieve the data item in the table. If Suprtool does NOT find an entry in the table, a zero will be returned if the data type is numeric and spaces will be returned if it is a byte type.

So, using the case above, if no entry is found in the table, zero is returned, and if int-field is equal to zero, then the record will qualify.

If you don’t want to have any values returned from the table lookup, you just preface the if with a standard lookup.

```
>get    ord-details
>table  cust-table, cust-no, file, custlist,data(state-code)
>if     $lookup(cust-table,cust-no) and &
       $lookup(cust-table, cust-no, state-code) = state-code
>output orders
>xeq
```

When using $lookup to return data, the $lookup must always be on the left side of the expression. If not Suprtool will stop with an error:

```
>if id-field=$lookup(mytable,char-field,id-field)
Error:  $lookup in this context (data comparison) must be on left side
```

Sorting Database Records

Sort Records

This extracts all records from the dataset d-sales into a data file called "result". The records are sorted by the field cust-account. Use the Sort command to do sorting.

```
>get    d-sales
>sort   cust-account                    {sort by cust-account}
>output result
>xeq
```

Sort Records in Descending Order

This extracts all records from the dataset d-sales into a data file called RESULT. The records are sorted by the field sales-total in descending order (i.e., show highest totals first). Use the Desc option in the Sort command to do this.

```
>get    d-sales
>sort   sales-total desc                {descending order}
>output result
>xeq
```

Sort by Multiple Keys

This extracts all records from the dataset d-sales into a data file called "result". The records are sorted by the field cust-account, then by sales-total in descending order. Use two Sort commands to do this, since the Sort command only accepts one field at a time.

```
>get    d-sales                          {input}
>sort   cust-account                     {first sort key}
>sort   sales-total desc                 {second sort key}
>output result                           {output}
>xeq
```

Deleting Records

To delete records safely from a database, you need two Suprtool tasks. The first task backs up the data to be deleted; the second task deletes the records. You can interrupt and restart the two-task method without harming your records. It may be faster to combine the two tasks, but it can present problems: if you abort the task part-way through, your database will be partially deleted, but you will not have a backup of the deleted records.

The first task copies the data to a backup file:

```
>get    d-sales
>if     trans-date>=000101 and trans-date<=001231
>output archives
>xeq
```

The second task deletes the entries:

```
>get    d-sales
>if     trans-date>=000101 and trans-date<=001231
>delete
>output $null
>xeq
```

Updating Records

To update records, you can specify the fields to be updated with the Extract command(s). The Update command must be entered before the Extract command. You can Update part of a field by defining a portion of the field with the define command and using the defined field in the Extract command.

You can update critical fields (IMAGE search or sort fields), by specifying the Ciupdate option on the Update command. CIUPDATE must be turned on or Allowed in the database. This can be turned on from within the DBUTIL utility.

You can also update using Arithmetic Expressions:

```
>get    d-sales
>update
>extract total = price * qty + tax
>xeq
```

You can update critical fields:

```
>get    d-sales
>update ciupdate
>if     product-no = 12345677
>ext    product-no = 12345678
>xeq
```

Duplicate Records

In the following examples, the key field is in the first four bytes of the record. "Duplicate-ness" is based on records having the same key value. In any group of records with the same key value, the first record is considered to be the "original", and the rest are considered to be the "duplicates".

Report without Duplicate Records

This is an example of filtering out duplicated records (the original remains). This is done by using the None option of the Duplicate command.

```
Input      Result

1111 a     1111 a

2222 b     2222 b

2222 c     3333 e

2222 d
```

```
3333 e
```

```
>get    dataset
>sort   keyfield
>dup    none keys
>output result
>xeq
```

Report Only the Duplicate Records

This is an example of keeping only the duplicated records (the original is not kept). This example is the opposite of the previous example. Use the Only option of the Duplicate command to do this.

```
Input       Result

1111 a      2222 c

2222 b      2222 d

2222 c

2222 d

3333 e
```

```
>get    dataset
>sort   keyfield
>dup    only keys
>output result
>xeq
```

Report Only the Unique Records

This example shows how to report only those records without duplicates. That is, if the records have duplicates, both the originals and their duplicates are omitted from the report.

```
Input        Result

1111  a      1111  a

2222  b      3333  e

2222  c

2222  d

3333  e
```

You have to use two Suprtool tasks to accomplish this. The first task creates an intermediate file Dupfile that contains the keys of the duplicate records. The second task creates the desired output file Result that contains only the unique records.

```
>get     dataset
>sort    keyfield
>extract keyfield
>dup     only keys
>output  dupfile
>xeq

>get     dataset
>table   dup-table, keyfield, sorted, dupfile
>if      not $lookup(dup-table, keyfield)
>output  result
>xeq
```

Report Only the Duplicates and Their Originals

This performs the opposite function to the one outlined above. It keeps only the duplicates and their originals.

```
Input       Result

1111  a     2222  b

2222  b     2222  c

2222  c     2222  d

2222  d

3333  e
```

Once again, you have to use two Suprtool tasks to accomplish this. The first task creates an intermediate file Dupfile that contains the keys of the duplicate records. The second task creates the output file Result that contains only duplicate files and their originals.

```
>get     dataset
>sort    keyfield
>extract keyfield
>dup     only keys
>output  dupfile
>xeq

>get     dataset
>table   dup-table, keyfield, sorted, dupfile
>if      $lookup(dup-table, keyfield)
>output  result
>xeq
```

Deleting Duplicate Database Records

The method shown here assumes that each record can be distinguished from each other, even though they have duplicate keys. You cannot use just one task to delete duplicates. To do the job safely, you need three tasks.

1. Identify which records to delete.
2. Archive those records.
3. Delete those records.

Task 1: Identify which records to delete.

```
>get    dataset
>define rec,1,length of your record
>sort   key1                                {sort enough keys to uniquely …}
>sort   key2                                {… identify all the duplicates}
>dup    only keys
>out    dupfile
>xeq
```

Task 2: Write records to archive.

```
>get    dataset
>table  duptab, rec, sorted, dupfile, hold
>if     $lookup(duptab, rec)
>output archive
>xeq
```

Task 3: Delete the records.

```
>get    dataset
>if     $lookup(duptab, rec)
>delete
>xeq
```

Deleting Non-Unique Duplicate Database Records

If you need to delete duplicates that cannot be uniquely identified (i.e., the entire record is a duplicate), use the following set of four tasks:

Task 1: Get a list of the duplicates.

```
>get    dataset
>define rec,1,length of your record
>sort   rec
>dup    only record
>out    dupfile
>xeq
```

```
Dataset   Dupfile   Dupfile2

a         a

a         a

a         c

b

c

c
```

Task 2: Get a unique list of the duplicates.

```
>input  dupfile
>dup    none record
>output dupfile2
>xeq
```

```
Dataset   Dupfile   Dupfile2

a         a         a

a         a         c

a         c

b

c

c
```

Task 3: Delete the duplicate records, including the original.

```
>get    dataset
>table  duptab, rec, sorted, dupfile2
>if     $lookup(duptab, rec)
>delete
>output archive
>xeq
```

```
Dataset   Dupfile   Dupfile2

b         a         a

          a         c

          c
```

Task 4: Restore one copy of the original duplicates.

```
>input  dupfile2
>put    dataset
>xeq
```

```
Dataset   Dupfile   Dupfile2

a         a         a

b         a         c

c         c
```

Deleting Duplicate Data File Records

This is a similar procedure to deleting database records. The main difference is that you use the Input and Output commands instead of the Get and Delete commands. You also need to define your sort fields.

Task 1: Identify which records to delete.

```
>input  datafile
>define key1,1,8
>define key2,13,12
>define rec,1,length of your record
>sort   key1
>sort   key2
>dup    only keys
>out    dupfile
>xeq
```

Task 2: Write records to archive.

```
>input  datafile
>table  duptab, rec, sorted, dupfile, hold
>if     $lookup(duptab, rec)
>output archive
>xeq
```

Task 3: Delete the records.

```
>input  datafile
>if     not $lookup(duptab, rec)
>output result
>xeq
```

Sorting IMAGE to KSAM

Here is how to extract all customer numbers from a master dataset, sort them, then write them to an existing KSAM file, erasing the KSAM's previous contents. Please note that this gives you sorted access to your IMAGE data if used properly.

```
>base store.data
Password >                              {password is not echoed}
>get m-customer
>sort cust-account
>out keycust,key,erase                  {keycust was built with KSAMUTIL}
>exit                                   {xeq implied, then leave Suprtool}
```

Loading a Dataset

To load records into an IMAGE dataset from a disc file, use these commands:

```
>base store.data                        {specify which database}
>input transfile                        {input is from a disc file}
>put d-sales                            {add each record to the D-SALES dataset}
>xeq                                    {execute the current task}
```

The layout of the input file must have the exact same format as the target.

Decimal Places

Data in files and databases often have an implied number of decimal places. For example, dollar amounts usually have two implied decimal places for the cents. In this case, the number stored is scaled by a factor of one hundred (e.g., you would enter 10000 to represent $100.00).

```
>get  d-sales
>if   total-sales > 99900               {find sales > $999}
>out  result
>xeq
```

You can use Suprtool's Item command to identify defined fields or database items that have an implied number of decimal places.

You can use Suprtool's Item command to identify defined fields or database items that have an implied number of decimal places. Once you do this, you can then enter regular, unscaled numbers. For example, to enter five cents, use 0.05; to enter $100.00, use 100. If a field is a dollar and cents amount scaled by 100, use the following to tell Suprtool about the decimal place:

```
>item total-sales, DECIMAL, 2
```

With the Item command, the previous example becomes more understandable:

```
>get  d-sales
>item total-sales, decimal, 2
>if   total-sales > 999                {find sales > $999}
>out  result
>xeq
```

Converting Numbers

There are several ways to convert binary numbers (e.g. I2, P8) into human-readable ASCII form. You can use STExport's Output,ASCII or Output,DISPLAY if you want to convert all numbers.

If you want to convert only some of your numeric fields, you can use Suprtool's numeric conversion feature to convert the binary fields to display fields.

```
define mynumber,1,6,display
get dataset
ext some-fields...
ext mynumber = binary-number
output filename
xeq
```

Note that this technique also works for converting a number from one numeric type to another numeric type.

You may also convert from binary to ASCII directly by using the $edit function.

```
define mynumber,1,6,byte
get dataset
ext some-fields...
ext mynumber = $edit(binary-number,"999999")
output filename
xeq
```

Counts and Subtotals

Count and Subtotal on Sort Keys

This example counts the number of sales transactions for each customer and produces the total sales for each customer. We use the Count and Total options of the Duplicate command. Note that we made the output file self-describing so we can easily work with it later.

```
>get  d-sales
>ext  cust-account
>sort cust-account                    {need to sort by key}
>item sales-total,decimal,2           {decimal places}
>dup  none keys count total sales-total
>list standard
>out  result, link
>xeq
```

The output file contains three fields. The first field is the cust-account that we extracted. Suprtool created two new fields at the end of each output record: st-count and st-total-1. St-count contains the number of times each cust-account occurred, while st-total-1 contains the total sales for each cust-account.

Sort by Count or Subtotal

When Suprtool counts or subtotals, the output is sorted according to the key fields. If you want your output file to be sorted by the count or by a total, you must process the output file with a second task. The following example sorts the previous file of totals by ST-COUNT. We choose a descending sort sequence, so that we can see first the customers with the largest number of orders.

```
>input result                          {input from previous task}
>sort  st-count, desc                  {highest counts appear first}
>list  standard                        {produce a simple report}
>xeq
```

Total by Field

If you want to get a single total for a field, without caring about subtotals on sort breaks, you can use the Total command. Total prints out the result on $stdlist. For example, to compute the total sales value for 2000 transactions, use these commands:

```
>get    d-sales
>if     purch-date>=000101 and purch-date<=001231
>total  sales-total
>output $null
>xeq
```

Running Totals

You can get a running total on a field using the $total function.. The target data must be a packed field with 28 digits, in order to help avoid overflow issues. A sample use of the total function could be:

```
  >def mytotal,1,14,packed
  >get orders
  >ext mytotal = $total(sales-amount)
  >xeq
```

Running SubTotals

Suprtool has the ability to keep a running subtotal for any numeric field based on a given sort key. The target data must be a packed field with 28 digits, in order to help avoid overflow issues.

A sample use of the $subtotal function could be:

```
  >def mytotal,1,14,packed
  >get orders
  >sort order-number
  >ext order-number
  >ext part-number
  >ext description
  >ext sales-amount
  >ext mytotal = $subtotal(sales-amount,order-number)
  >out sales,link
  >xeq
```

This would result in a file containing a running subtotal in the field mytotal for a given order-number. You could then generate a simple report with the simple Suprtool commands:

```
>in sales
>list standard
>xeq
```

The basic syntax for the $subtotal function in the extract command is:

```
extract targetfield = $subtotal(field,sort-field)
```

You must specify the sort command before referencing the sort-field in the $subtotal function. You can subtotal up to ten fields per task.

Listing Records

You can print selected input records either formatted or with the Octal, Hex, Decimal, or Character representations. To dump all sales records with a negative amount, use these commands:

```
>get d-sales
>if sales-total < 0
>list lp
>xeq
```

This finds the entries that meet the selection criteria and prints them to the default line printer, showing field names and field-values converted to ASCII. If you suspect that your data is bad, you can dump the records in Octal/Char format instead:

```
>get d-sales
>if sales-total < 0
>list octal,char
>xeq
```

If you want the listing in column format, use List Standard:

```
>get d-sales
>if sales-total < 0
>list standard lp
>xeq
```

Changing the Output Record Format

You can change the output file record format by using the Extract command. The Extract command causes Suprtool to assemble Output records by stringing together fields extracted from Input records. You would use the following to extract two of the nine fields from the customer records:

```
>get m-customer                 {input from a dataset}
>extract cust-account           {extract the key value and …}
>extract credit-rating          {… one other field}
>output out1                    {output file will have two fields}
>xeq
```

You can easily insert data into the middle of a record, again using the Extract command. Define the first and second halves of the record as two big chunks. Now Extract the first part, note the constant you wish to insert, then Extract the second part.

```
>input myfile                              {95 bytes wide}
>define part1,1,60,byte                   {first 60 bytes}
>define part2,61,35,byte                  {remaining 35}
>extract part1, "constant", part2         {extract an 8-byte constant}
>output newfile                           {103 bytes now}
>xeq
```

Producing a Condensed Dataset Listing

When debugging test databases, it is often desirable to produce a condensed listing of a dataset on $stdlist. The following example combines the Extract command with the ASCII output option (i.e., all binary and packed-decimal data is converted into readable ASCII characters). For readability, each data value is prefixed with an abbreviated field name. This listing is more compact than the one produced by the List command.

```
>get m-customer
>extract "Account=",cust-account," "
>extract "C/R=",credit-rating
>output *,ascii                           {* implies $stdlist}
>xeq
```

The output would look like:

```
Account=04598921 C/R=    500000
Account=44657844 C/R=   2000000
Account=98753198 C/R=    300000
```

Simple Reports

You can produce simple reports with Suprtool's List command. You select the records for the report with the If command and the fields for the report with the Extract command. Reports can include running headings with the date, title, and page number and an optional line of column headings. Suprtool can produce default titles and headings.

```
>get     m-customer
>item    credit-rating,decimal,2
>extract cust-account
>extract credit-rating
>list    standard
>xeq
```

The output would look like this:

```
Feb 16, 2000     Base STORE Set M-CUSTOMER     Page 1
CUST-ACCO CREDIT-RATING

 4598921       5000.00
44657844      20000.00
98753198       3000.00
```

Your First Report

Our report selects all customers in California, sorts the records by city, and reports on the city, account number, and name of each California customer:

```
>get     m-customer                    {input dataset}
>if      state = "CA"                  {California customers}
>sort    city                          {sort by city name}
>extract city                          {city first on each line}
>extract cust-account                  {followed by account#}
>extract name-first                    {and first name}
>extract name-last                     {and finally last name}
>list    standard                      {produce a quick report}
>xeq
```

These commands produce a report with four columns. The title consists of the Store database and the m-customer dataset. The column headings are the name of each field that we extracted.

Printing a Report

By default, the List command writes output to $stdlist, where lines are folded as necessary to fit within 78 columns (to prevent line overflow).

To write the report to a system line printer, use the following :file command before using the List command:

```
>:file suprlist;dev=lp;rec=-132
```

If you are writing a report to a LaserJet, we suggest that you add this command to your task. It selects 175 column output in Landscape mode.

```
>set list pcl 1
```

Specifying Your Own Title

If you don't like Suprtool's default title, you can override it with your own:

```
>list standard,title "Customers in California"
```

Column Headings

Column headings default to uppercase field names. The names are truncated if they are longer than the field itself. One space is inserted between fields.

Suprtool does not automatically align user-specified headings with the data columns. We suggest specifying heading strings with the same length as the fields they represent, while taking into account the space between the data columns.

In our example, we enter one column heading per line (using Suprtool's continuation character "&"):

```
>list standard,heading  &
      {----+----1----+----2}
      "City          " &                {field is X12}
      "Account    "    &                {field is Z8}
      "First Name "    &                {field is X10}
      "Last Name"                       {field is X16}
```

We included one space between fields. Note that an extra space was needed for the Account heading (it is an 8-digit field, but we used 10 characters). Because cust-account is a zoned-decimal field, an extra space is required for the sign.

Printing Mailing Labels

You can print mailing labels by combining the Extract command with the List Oneperline command. We assume that each mailing label starts with two blank lines, followed by the customer name and address, followed by another blank line. The Suprtool commands to produce the labels are as follows:

```
>get     m-customer                          {input customers}
>extract " "                                 {first field}
>extract " "                                 {second field}
>extract customer-name                       {name first}
>extract street-address(1)                   {three lines of address}
>extract street-address(2)
>extract street-address(3)
>extract " "                                 {last blank field}
>list    oneperline, noname, noskip, norec
>xeq
```

The extract " " line creates a single field that consists of a blank space. Each of these blank fields results in a blank line on our mailing labels, since the List command puts one field on each output line.

If you want to combine two fields on one line, you would first have to create an output file with the combined fields and use this file as input to List Oneperline.

Running Suprtool under MPE

How to Run Suprtool

To access Suprtool, type the "run" command:

```
:run suprtool.pub.robelle

SUPRTOOL/Copyright Robelle Solutions Technology Inc. 1981-2001.
(Version 5.9)  SUN, JUN 17, 2001, 11:47 AM  Type H for help.
>
```

Suprtool prints its version number and the current time right after a banner. It also prints the name of the company that has licensed this copy of Suprtool.

Suprtool then prompts with ">". Press Return after typing each command. For example, if you type the help command:

```
>help
```

Suprtool prints some help text and a keyword list. Type a keyword or press Return to leave Help.

How to Xeq a Suprtool Task

Normally, you enter a series of commands. These commands specify the input source, the output destination, and any special processing that is required. Finally, you enter an **Xeq** or an **Exit** command. This begins the actual Suprtool extraction task.

If you entered the Exit command, Suprtool finishes the current task, then returns you to MPE. If you entered the Xeq command, Suprtool finishes the current task, then prompts you for another task. This continues until you enter the Exit command. If you wish to terminate Suprtool immediately (perhaps you are confused), enter Exit Abort, which immediately returns you to MPE without attempting any task. If you :Run Suprtool from another process (e.g., Qedit), you can suspend execution without executing the current task with the Exit Suspend option.

Info= for Commands

If you run Suprtool with Parm=4, Suprtool processes up to 80 characters of command input specified via Info=.

```
:run suprtool.pub.robelle;parm=4;info="I ABC;O ABC2;E"
```

Suprtool terminates only if the Info= string contains an E command. Otherwise, it prompts for more commands after processing the Info= command string. A warning is printed if you accidentally specify an Info= string longer than 80 characters.

You may want to combine one of the more useful Suprtool commands with the Info= option to standardize certain Suprtool operations. One restriction of the Info= string is that it may only be 80 characters long. You may need to have many Suprtool commands executed. For example, suppose that the file SUPRUSE contained commands to set up a standard Suprtool extract, but the selection criteria changed with each use. Invoke Suprtool with:

```
:run suprtool.pub.robelle;parm=4;info="USE SUPRUSE"
```

The user can enter the If command and an Exit command, all of the other options could be in the Supruse file. An advantage of this method over using :RUN SUPRTOOL;STDIN=XXX, is that the input file Supruse can be a Qedit file. With the STDIN= option, the file must be an unnumbered /Keep file.

Son Process

If you run Suprtool within Qedit or SELECT, you can retain the Suprtool process and re-activate quickly later.

```
:run qedit.pub.robelle
/:r suprtool.pub.robelle
>...                                    {suprtool commands}
>exit
SUPRTOOL.PUB.ROBELLE is still alive. Okay to HOLD ? Y
/...                                    {qedit commands}
/:activate
SUPRTOOL.PUB.ROBELLE
>...                                    {more suprtool commands}
>exit
Program Held.  Use :ACTIVATE/:RUN;HOLD to re-run.
/...                                    {more qedit commands}
```

Combining Info= and Son Process

Normally, when you re-activate Suprtool from a HELD state, Suprtool ignores the original Info= string. However, if you run Suprtool with Parm=8, its re-processes the Info= commands every time you activate it. One application of Parm=8 is with usefiles. You can hold Suprtool suspended and have it execute a usefile each time you awaken it. By changing the usefile between activations, you can make Suprtool do any task you like:

```
:run qedit.pub.robelle
/... edit file named DOIT, of suprtool commands
/:run suprtool.pub.robelle;parm=8;info="USE DOIT"
 ... suprtool executes commands from DOIT
>exit                                   {return to Qedit}
/... edit DOIT
/:activate                              {returns to Suprtool, executes Info= again}
... executes new commands in DOIT
```

Exit with Verify

Some users find that they Exit from Suprtool inadvertently. To require Suprtool to get user approval on Exit, run Suprtool with Parm=64.

```
:run suprtool.pub.robelle;parm=64
>e
Okay to exit [no]:
>
```

Preventing Suprtool from Suspending

If you run Suprtool from within HPDesk (and some other programs), Suprtool suspends on Exit but HPDesk does not notice. The next time you run Suprtool you will get a new copy of Suprtool. Eventually you will have many suspended copies of Suprtool hanging from HPDesk, consuming system resources. Turning Set Suspend Off forces Suprtool to terminate on Exit rather than suspend. You can force Set Suspend Off by running Suprtool with Parm=32.

```
:run suprtool.pub.robelle;parm=32
```

Stdin and Stdlist Files

Suprtool reads command lines from $stdinx, the MPE predefined name for the standard job input file. This is usually the terminal or the stream file. You can redirect the input by using an option of the :Run command. For example:

```
:comment  kcommand created with editor
:run suprtool.pub.robelle;stdin=kcommand
```

You can also use the I/O redirection capabilities of MPE/iX to achieve the same goal.

```
:run suprtool.pub.robelle<kcommand
```

This file must not be numbered; use /Keep xxx,unn in EDIT/3000 or use /KQ in Qedit. The maximum record width of this input file is 256 characters. Note: the Stdin file must not be a Qedit workfile.

Suprtool prints all ASCII output messages to $stdlist, the MPE predefined name for the standard job listing file. This is usually the terminal or the line printer, but it can be re-directed with the :Run command. For example:

```
:comment  throw away the output from suprtool.
:run suprtool.pub.robelle;stdlist=$null.
```

Input and Output Files

As with SORT/3000, it is possible to specify both the input file and the output file using :File commands, and then specify only the Key fields as Suprtool commands. Only one task is allowed per RUN with this method:

```
:file input=oldxx
:file output=newxx
:run suprtool.pub.robelle
Suprtool/Copyright Robelle Solutions Technology Inc. 1981-2001
(Version 6.1)
>key 1,10
>key 20,5
>exit
```

Using CREATEPROCESS

Since both command input and message output are under user control, it is possible to run Suprtool as a "son process" from within an application program (see the MPE Manual for CREATEPROCESS). You can check the Job Control Word to see if Suprtool called Quit and terminated in an error state. See the Suprcall user manual for a complete description of a working interface that is available to all Suprtool users.

Duplicating Files

One of the most common reasons for using Suprtool in a SESSION is to duplicate a disc file quickly. Accordingly, Suprtool has a special option to facilitate this function. If you :RUN the Suprtool program with Parm=16, a file copy is implied from Input to Output and no commands are needed. :File commands are used to specify the names of the files. For example:

```
:file input = xxx
:file output = xxxnew
:run suprtool.pub.robelle;parm=16
:reset input
:reset output
```

With the User Defined Command facility (UDC) of MPE, you can reduce all of this to a single command,

```
:dup xxx,xxxnew.
```

The Info= parameter is ignored in this mode.

Note that the MPE/iX Copy command is faster than Suprtool for simple file duplication.

Job Control Word

Suprtool sets the system job control word JCW to a fatal state when Suprtool fails in a batch job. Suprtool sets only the high-order bit of the JCW job control word. That is, it adds 32,768 to the existing JCW value. HP subsystems use the other bits of the JCW job control word, so Suprtool does nothing to them.

Suprmgr Configuration File

Each time that you run Suprtool, Suprtool attempts to open and "use" a disc file named Suprmgr in Pub.Sys. It does this to check whether the System Manager has specified any default configuration values. The Suprmgr file may be an Editor /Keep file,UNN or a Qedit workfile. No more than 256 characters per record are processed.

Because Suprtool treats the Suprmgr file exactly like a usefile, Suprmgr can include any Suprtool commands. Use Set commands in the Suprmgr file to establish custom defaults for the shop.

Dbedit Set values can also be changed from within the Suprmgr file. You must use the Edit command, enter the Dbedit Set commands, and then Exit back to Suprtool. If you forget the Exit, each execution of Suprtool leaves you in Dbedit. An example Suprmgr file might be:

```
set dumponerror off                {don't print error records}
set varsub on                      {enable variable substitution}
edit                               {enter Dbedit}
set quiet on                       {print shorter messages}
exit                               {return to Suprtool}
```

Using Suprtool in Batch

Suprtool operates in "session" mode or "batch" mode. In batch usage, any "error" message causes Suprtool to quit, setting the Job Control Word to flush the remainder of the JOB. Warning messages do not cause an abort.

In batch mode, Suprtool does not prompt for missing information as it does in session mode. Instead, it attempts to choose the alternative that has the least chance of destroying valid data. For example, if the output file is a duplicate file name in batch mode, Suprtool saves the new output file with a "made up" name (Output*nn*, where nn is from 00 to 20), prints a warning message, and aborts. Another example is that of missing database passwords. In session mode, Suprtool prompts for the password; in batch mode, it uses the CREATOR password instead.

SuprtoolOutCount JCW

When Suprtool closes the output file, it sets a JCW named SuprtoolOutCount with the number of output records. This is the same number reported in the total line (i.e., OUT=). You can use this JCW to control job stream execution by checking if the SuprtoolOutCount JCW is non-zero. If there are more than 65,535 output records, Suprtool sets SuprtoolOutCount to 65,535.

You can use the :Showjcw command to see the value of a JCW. For values greater than 16K, Showjcw displays either the word WARN, FATAL or SYSTEM followed by some digits. These words correspond to the following values:

Word	Value
WARN	16384
FATAL	32768
SYSTEM	49152

Add the value of the word to the number that appears with it for the true value of SuprtoolOutCount. For example,

```
:showjcw suprtooloutcount
SUPRTOOLOUTCOUNT = WARN8616
```

16384 + 8616 = 25000

The MPE/iX :Showvar command can also be used to see the value of a JCW. Showvar displays the full, correct number (e.g., 25,000) up to the maximum of 65,535.

Suprtool also sets two other JCWs: SuprtoolOutCount1 and SuprtoolOutCount2. These communicate the full OUT= value to the Suprtool2 interface.

SuprtoolFullCount Variable

On MPE/iX, when Suprtool closes the output file, it also sets a variable named SuprtoolFullCount with the number of output records. This is the same number reported in the total line (i.e., OUT=). The advantage of the variable is that if more than 65,535 records are written to the output file, the value of the SuprtoolFullCount variable is not truncated.

Summary of Parm= Values

Value	Meaning
4	execute Info string once
8	execute Info string on each re-activation
16	duplicate Input file to Output file
32	don't suspend, terminate completely
64	check with user before Exiting

Values may be combined by adding them together. For example, Parm = 96 means "check with me before exiting, then when I do actually exit, terminate Suprtool completely instead of suspending".

Suprtool Issues and Solutions

A Suprtool Task

Suprtool's primary function is to extract data quickly; its focus is batch extraction. The key principle is: *the bigger the input data source and the smaller the subset of data selected; the more performance improves.*

Your aim is to replace serial reads and selection with Suprtool. To do this, break your task into components: an *input choice*, some *processing* selections, and the *output choice*.

Input Choices

Suprtool can read input from an MPE or KSAM file, an IMAGE dataset, or a tape file. Using :File commands, any MPE file with fixed-length records can be read by Suprtool. Your first action is usually to select a database with the Base command. Then, use the Chain or Get command for input from a dataset, and the Input command for all other input choices.

Often you select a subset of the input records using the If command. Only selected records are passed to the processing stage and the output choice.

Processing Selections

If you do not specify any processing, the input records are quickly copied to the output choice. Some of your processing choices are

1. Sort the records into ascending or descending sequence (Sort or Key). No records are output until all of the selected input records have been sorted.
2. Total one or more input record fields (Total).
3. Remove or select duplicate records (Duplicate).
4. Delete the records from an IMAGE dataset (Delete).

Output Choices

Usually you wish to extract a subset of your records to feed into a report program, so the default output file is a data file. The default output file format matches the input file format, unless you use the Extract command. The output choice can be an MPE or KSAM file, an IMAGE dataset, or a tape file. You can use :File commands to create almost any fixed-length MPE file from Suprtool (including $stdlist). You can

specify different formats for the output file by qualifying the Output command. To have readable ASCII output, use "output xxx,ascii". To produce "self-describing" files, use Output xxx,Link.

By default, every output record is identical to the corresponding input record. The Extract command assembles output records by stringing together fields extracted from the input records. With the Extract command you can insert constant values into the output record.

Each output record is written to the output choice. You can also see a formatted listing of each record with the List command. Records can be added to an IMAGE dataset just before being written to the output file (use the Put command). By default, Suprtool builds a new file, unless you specify Append or Erase. You would use Erase to write to a Quiz subfile (for details, see "Suprtool and PowerHouse Applications" on page 83).

Suprtool and Allbase

Specify an Allbase database with the **Open** command. Once Suprtool has opened the database, use the Form command to obtain information about the tables in the database. Use the Select command to choose what data to read from your Allbase database.

We have tested the Allbase module with Allbase version G.1.13 for MPE. We believe that it will be compatible with future versions of Allbase. We have not tested Suprtool with any of the F versions of Allbase. Allbase access is available as a separate add-on module to Suprtool.

Data-Types

When you specify a Select command, Suprtool figures out how to translate the Allbase internal data-types into ones that Suprtool can process. Not all Allbase data-types can be processed by Suprtool. The following table lists the Suprtool data-type that corresponds to the Allbase data-type:

Allbase Data-Type	Suprtool Data-type
Integer	Double
Smallint	Integer
Binary	Not Supported
Char	Byte
Varchar	Byte
Real	Ieee-32
Float	Ieee-64
Decimal	Packed
Numeric	Packed
TID	Not Supported
Date	Byte
Time	Byte
Datetime	Byte
Interval	Byte
Varbinary	Not Supported

Long binary	Not Supported
Long varbinary	Not Supported

Date and Time Types

Allbase has four types of fields that are associated with dates and times. These fields are converted to byte-type data and are returned with specific lengths.

The date and time fields are returned with the following byte lengths:

Data-Type	Length
Date	10
Time	8
Datetime	23
Interval	8

Restrictions

Suprtool cannot currently handle all Allbase database concepts. The current restrictions are:

1. Suprtool requires that the ownername be specified when selecting a particular table in the following manner:

   ```
   >select * from purchdb.orders
   ```

   In this example the owner is purchdb and the tablename is orders.

2. Suprtool does not currently handle the Allbase date format. You can convert the Allbase date format to something that Suprtool can handle with the TO_CHAR function in the Select statement. For example:

```
>select qty,TO_CHAR(date,'YYYYMMDD') from manufdb.testdata
>def mydate,date[1],8                    {redefine testdate }
>item mydate,date,yyyymmdd               {define the date format}
>if mydate<=$today(-900)
```

Suprtool and IMAGE

You specify an IMAGE database with the **Base** command. Once Suprtool has opened the database, all IMAGE item, set, and field information is available. To identify an IMAGE item as a date-field, use the Item command. Suprtool has facilities to Get, Put, and Delete database records.

Reading Datasets

For Suprtool to read a dataset, it must have read-access to every field in the dataset. If you have a database with item-level security, you must be careful to choose a database password that gives access to all of the fields in the dataset. The Creator password *always* gives you full-read access to every dataset.

By default, Suprtool opens databases in mode-1. You may use any other mode except mode-7. If you use the Delete or Put commands, you must open the database with a mode that permits changes to the database. If Set Defer On has been specified, the database must be opened in mode-3.

Serial vs. Keyed Access

The Get command provides serial access to a dataset; for chained access use the Chain command. This leads to some results that seem obvious but are easily overlooked. If you specify input with the Get command and sort an IMAGE dataset by a search field value such as custnum, Suprtool does a serial scan of the entire dataset. It does not read down the chains for each custnum value. Therefore, if the resulting sort file contains entries with identical sort values, they are sorted in the order they were found in the serial scan, not in the order they had on the chains. It is good practice to ensure that implicitly sorted paths can also be explicitly sorted, without benefit of the IMAGE chains. If you specify date as a sort field, don't overlook the possibility of two entries with the same custnum and date values. Do they have an implied order, perhaps by trans-type?

When to Use Suprtool with IMAGE

Suprtool is a solution for those programs that serially read and select data from large IMAGE datasets. Suprtool performs best when you are selecting less than 50% of the entries from the dataset (also see the **Speed Demon** User Manual). To replace a serial scan of a dataset, use the following steps:

1. Use the Get and If commands of Suprtool to scan and select your entries.
2. Optionally, sort the records with Suprtool.
3. Use the Output command to put the selected records in an MPE file.
4. Change the application program to read the MPE file instead of the IMAGE dataset. Each record in the file has exactly the same layout as a record from the dataset.

Suppose that your program scans the d-sales dataset to produce a report of all d-sales records with a sales-total greater than $10,000.00. Currently, your program looks like this:

```
10-read-and-report         section.
    perform 10-10-get-d-sales
    if not end-of-d-sales then
        if dsa-sales-total > 10000 then
            perform 20-report-d-sales
               thru 20-report-d-sales-exit.

    go to 10-read-and-report-exit.

10-10-get-d-sales.
    *Use DBGET to read a d-sales record into buffer-d-sales.*

10-read-and-report-exit.  exit.
```

Assuming that only a small percentage of the d-sales dataset have a sales-total greater than $10,000.00, you would replace this scan with the following:

```
:run suprtool.pub.robelle
>base store,5                          {open for read access}
>get d-sales                           {scan the d-sales dataset}
>if sales-tot > 1000000                {note implied decimal}
>output dsales                         {output to an MPE file}
>exit
```

You would modify your application program to read the dsales MPE file:

```
10-read-and-report         section.
    read sales-file into buffer-d-sales at end
        move true to end-of-sales-flag.

    if not end-of-sales then
        perform 20-report-d-sales
           thru 20-report-d-sales-exit.

10-read-and-report.  exit.
```

TurboIMAGE Compatibility

Suprtool is compatible with TurboIMAGE and TurboIMAGE/XL. Suprtool automatically adjusts for a Turbo database; you do not need different versions of Suprtool for different versions of IMAGE. Suprtool handles all of the limits introduced with TurboIMAGE.

Locking of IMAGE Datasets

Suprtool calls DBDELETE, DBPUT, and DBUPDATE for the Delete, Put, and Update commands. If the database is not open in mode-3 or mode-4, Suprtool must lock the dataset when adding or deleting dataset records. The locking strategy is determined by the value of Set Lock.

Locking for the Entire Task

Specifying Set Lock 0 forces Suprtool to lock either the Put and Delete datasets or the Update dataset for the entire task. The lock is done after the Xeq command. Suprtool uses a lock descriptor and one call to DBLOCK to lock two datasets in the same database. If both a Delete and a Put were specified to different databases, Suprtool locks them both, first the input dataset in one database, and then the output dataset in the second base. After the task finishes, the Put and Delete datasets or the Update dataset are unlocked.

Set Lock 0 provides the best performance, but other users cannot change the database during the entire Suprtool task. The reason for improved performance is twofold:

1. The database only needs to be locked once.
2. If Suprtool is reading the database with DBGET, it does not have to reget records and ensure that they have not changed with each lock.

Locking for Each Transaction

Set Lock 1 (the default), causes Suprtool to lock and unlock around every DBDELETE, DBPUT, and DBUPDATE transaction. This provides the least contention with other users of the database but may not be the best choice for large overnight runs where exclusive access to the database is available.

Locking Once in a While

Any other value of Set Lock causes Suprtool to lock and unlock the datasets every *n* transaction, where *n* is the value of Set Lock. A transaction is either a DBDELETE, DBPUT, or DBUPDATE -- it is not the number of input records read. After the Xeq command, Suprtool does not lock the datasets until the first DBDELETE, DBPUT, or DBUPDATE. After unlocking the datasets, Suprtool does not do another lock until a further DBDELETE, DBPUT, or DBUPDATE.

Record Changed Before It Could Be Deleted

If you are deleting or updating records when Set Lock is not zero, it is possible that a record will be modified, removed, or a different record added in its place while Suprtool has the database unlocked. Suprtool checks for this by retrieving the database record once the database is locked and checking it against the original record read. If any changes have occurred, Suprtool reports an error. If Set Ignore is Off, Suprtool stops the current task when the error occurs. If Set Ignore is On, Suprtool displays the records that were changed (with Set Dumponerror On) and continues processing.

Control-Y with Locks and Dbedit Locking

If you type Control-Y during a Put, Delete, or Update, the dataset locks are still in effect. Do NOT walk away from your terminal without answering YES or NO to the "stop" question. The Edit command uses its own locking strategy. For complete details on database locking with Dbedit, see the Dbedit manual.

Database Maintenance (Delete, Put)

The Delete and If commands of Suprtool give you a powerful facility for archiving data without writing COBOL programs. Be careful when using the Delete and Put commands. It is better to break up a Suprtool "run" into multiple passes in order to make the job more fail-safe. For example, we might like to archive all of the 2000 transaction records to a disc file and have them deleted from the database. The obvious way to achieve this is

```
:run suprtool.pub.robelle
>base actrec.data
>get  artrans
>if trans-date>=000101 and trans-date<=001231
>delete
>sort trans-date; sort cust-number
>output archives
>exit
```

What happens if this "run" is aborted part way through? You must :Restore your database and do a recovery. Any aborts will leave the database partially deleted.

If we break up this operation into two parts, archive first and delete later, we can easily restart the job in the case of an abort.

First, we read and sort the data into the archive file:

```
:run suprtool.pub.robelle
>base actrec.data
>get artrans
>if trans-date>=000101 and trans-date<=001231
>sort trans-date; sort cust-number
>output archives
>xeq
```

Second, we read and delete the entries. This method takes slightly longer, but it can *always* be restarted:

```
>get artrans
>if trans-date>=000101 and trans-date<=001231
>delete
>output $null
>exit
```

Warning: there is a window of vulnerability here -- if any qualified records are deleted or added by other jobs or sessions, the Delete pass may not delete the same records as written to the file in the first pass. In many applications, you might be able to use the extract file from the first pass to load a table in the second pass, then use Chain or If $lookup to delete the exact same records.

B-trees

HP has provided a generic search and partial key retrieval feature to IMAGE/SQL by attaching B-trees to a master dataset key item. This feature is available in versions of IMAGE/SQL C.07.03 and later. To determine the version number of IMAGE, run Query (:Run Query.Pub.Sys) and enter the Version command.

```
:run query.pub.sys
>version
```

Similarly, you can also run Dbutil.Pub.Sys, type Help, and look for the Addindex command.

B-trees are built only on the key item of a manual or automatic master dataset. However, if the path into a given detail dataset has an attached B-tree, you can implicitly take advantage of the B-tree by using a partial-key lookup on the associated detail.

Suprtool can take advantage of these B-trees via the Chain command. Suprtool also shows any B-trees that it can utilize through the Form command.

Master Dataset Expansion (MDX)

In newer version of IMAGE/SQL, master datasets can expand while records are being put into the database. This new capability is similar to the Detail Dataset Expansion (DDX), which was made available with MPE/iX 5.0.

Details on a dataset with MDX enabled are documented in the Form Command.

Jumbo Datasets and Large Datasets

On newer versions of MPE/iX, IMAGE/SQL supports Jumbo datasets, which can store data in one or more MPE files. By spanning files, you can have more than 4Gb of data in a single dataset.

In versions of Image prior to version C.07.14 the maximum size of a Jumbo dataset is 40Gb.

The maximum size of a Jumbo dataset was increased to 80Gb in IMAGE version C.07.14 and later.

In the C.09.02 version of Image, and later, the limitation is now based on 2 billion entries.

Limitations

Due to file system limitations, Suprtool can extract and sort more than 4Gb of data only on MPE/iX 6.5 and higher. In prior versions of MPE/iX Suprtool is limited to 4Gb. Due to limitations in MPE/iX Suprtool cannot open a remote Jumbo dataset.

Large File DataSets

Large File DataSets or LFDS are Image datasets that are single files which can be greater than 4Gb. Due to issues with the implementation of LFDS, Suprtool does not support them specifically, nor does it refuse to read them. We do not recommend using Large File DataSets nor will we support them.

Notes

If Set Stat is On and Set Prefetch is greater than zero, then each time a file in a jumbo dataset is closed, you receive a prefetch summary for each file. This could appear in the middle of a >List, if you are listing to $stdlist.

Suprtool and KSAM Files

Suprtool reads and writes both compatibility mode and native mode KSAM files. One exception is variable-length records, which Suprtool cannot handle. Use the Input command to read from a KSAM file. Use the Erase or Append option of the Output command to write selected input records to a KSAM file. When writing to a KSAM file, the input source can be an IMAGE dataset, a KSAM file, or an MPE file. You must build the KSAM file first as Suprtool, unlike FCOPY, cannot create new KSAM files.

Suprtool uses different methods for accessing compatibility-mode KSAM files versus native mode KSAM files. In our discussion of KSAM files, we designate compatibility-mode KSAM files by KSAM/V and native mode KSAM files by KSAM/XL.

Locking Output KSAM Files

Access to KSAM files is controlled with :File commands. After opening an output KSAM file, Suprtool checks to see what access was granted. If the KSAM file has share and lock access, Suprtool must lock the output file whenever it writes an output record. This provides the least contention with other users, but it slows down Suprtool (one lock per write).

Locking Input KSAM Files

By default, Suprtool opens Input KSAM (/V and /XL) files with lock-access for application reasons. We assume that application programs will likely want to write to KSAM files, and in order to write to any KSAM file the file must be opened with locking.

To override this default, you can specify the ;**nolock** option on a file command:

```
>:file ksamin;nolock
>input ksamin
>output mycopy
>xeq
```

Compatibility Mode KSAM/V Files

Suprtool uses high-speed NOBUF/MR access to quickly read input KSAM/V files. Suprtool is typically five times faster than FCOPY when reading KSAM/V files. Suprtool writes output records to KSAM/V files one record at a time. This ensures that the KSAM/V key information is updated correctly. Suprtool runs at the same speed as FCOPY when writing to KSAM/V files.

Suprtool must always have Shared, Lock, and Read access to all KSAM/V files. Suprtool briefly locks the KSAM/V file to check the physical EOF. If the EOF is wrong, Suprtool reopens the file with Locking and Inout access, in order to update the physical EOF. After checking and updating the physical EOF, Suprtool unlocks the KSAM/V file and leaves it unlocked while reading the file.

You should be aware of the following:

1. A user program that opens the KSAM/V file without Locking fails if Suprtool has the file open. You should specify Locking on all KSAM/V opens, even if you do not plan to lock the file.
2. A user program that opens the KSAM/V file with exclusive access will fail if Suprtool has the file open.
3. Suprtool cannot read KSAM/V files that user programs have open with exclusive access.

Suprtool ignores the KSAM/V key file; it reads the data file *only*. If a data record has a binary -1 in the first two bytes, Suprtool ignores the record. Your user programs *must* guarantee that a binary -1 does not appear in the first two bytes of any record, except those that have actually been deleted. If you wish to read the deleted records, use Set Recover On.

Reorganizing KSAM/V Files

Suprtool can be used to reorganize an existing KSAM/V file. Assuming that you have a KSAM/V file with a 10-byte key starting on the third byte, the following Suprtool commands sort the file along the primary key, remove any deleted records, and leave the KSAM/V file in an optimal state. Warning: OUTPUT =INPUT *cannot* be restarted if it is aborted.

```
:run suprtool.pub.robelle
>in ksamdata
>key 3,10,byte
>output =input
>exit
```

The above Suprtool run is dangerous because it cannot be restarted if it is aborted for any reason. The following Suprtool run can be restarted, even if the run is aborted.

```
:purge tempfile
:run suprtool.pub.robelle
>in ksamdata
>out tempfile
>xeq
>in tempfile
>key 3,10,byte
>output ksamdata,erase
>exit
```

KSAM/XL Files

Starting with MPE/iX 4.0, HP provided a new mechanism for tools like Suprtool to access KSAM/XL files. This mechanism allows Suprtool to work the same for either KSAM/V or KSAM/XL files. Some of the advantages are:

1. Suprtool is fifteen times faster than tools like FCOPY when reading a KSAM/XL file.
2. Suprtool reads the input file in chronological order.
3. Set Recover On works for KSAM/XL. Note that if you build the KSAM/XL file with the reuse option, the space from your deleted records may be reused. In this case, Suprtool may not be able to recover your records.

Suprtool will revert to slower method of reading KSAM/XL files if using selection criteria, in other words an if command.

Loading KSAM/XL Files

If you are writing records to a KSAM/XL file, you can dramatically improve the speed of Suprtool by writing to a temporary KSAM/XL file and saving it at the end of the task. Temporary files are not attached to the MPE/iX transaction manager, which results in faster writes. If the system fails while Suprtool is writing records to the temporary KSAM/XL file, the temporary file is lost. You want to ensure that you can safely restart the task if the system fails (e.g., you don't want to copy records to a temporary KSAM/XL file as you delete them from a dataset).

```
:build myksam;temp;rec=-128,1,f,ascii;disc=40000;&
:   ksamxl;key=(b,1,10,dup)
:run suprtool.pub.robelle
>input file.data
>output myksam,temp,erase
>xeq
>exit
:save myksam
```

How Do I Convert CM Ksam to NM KSAM?

Suprtool can't create a new NM KSAM file for you, but you can create the new NM file first using the MPE BUILD command (or some other method, see below), then use Suprtool to copy the records from the old file to the new file.

But Suprtool should definitely be able to copy the data *much* faster than MPE's FCOPY utility could.

The trick, of course, is getting a new NM KSAM file built correctly first. You could use a complicated BUILD command, or you could do this:

```
:file n=newfile; disc=
:fcopy from=oldfile; to=(*n); subset=0,1
```

That will create a new NM KSAM file called NEWFILE with the FLIMIT equal to the FLIMIT of your current CM file, and with all the keys set up correctly, but will copy only a single record into it. Then use Suprtool:

```
:suprtool
>in oldfile
>out newfile, erase
>xeq
```

Finally, if it's a big KSAM file you may wish to create it as temporary first, then SAVE it after you're done, to squeeze even more speed out of the process:

```
:file n=newfile; disc=; temp <<-- Note the temp designation
:fcopy from=oldfile; to=(*n); subset=0,1
:suprtool
>in oldfile
>out newfile, erase
>exit
:save newfile
```

Reuse option and KSAM/XL Files

The Reuse option forces deleted record space to be reused in KSAM files. In order to take advantage of the Reuse option in KSAM/XL files you must specify this option with a file command.

```
>input file.data
>:file myksam;reuse
>output myksam,append
>exit
```

Suprtool and MPE Files

Suprtool reads MPE files as easily as IMAGE datasets, with the single exception of variable-length records, which Suprtool cannot handle. Instead of using the Get or Chain command to specify your input source, use the Input command to read from an MPE file. The default output file is an MPE disc file named "OUTPUT", but you can control the name and format of that file with the Output command and the MPE :File command.

Buffered and NOBUF File Access

You can control the way Suprtool accesses MPE files through the use of :File commands. Normally, Suprtool tries to read and write MPE files with NOBUF and multirecord access. To force Suprtool to read one *block* at a time, you would use:

```
>:file file.input;nomr
>input file.input
>output file.output
>xeq
```

To force Suprtool to write one *block* at a time, you would use:

```
>input file.input
>:file file.output;nomr
>output file.output
>xeq
```

If your files have large block sizes or Suprtool seems to read the file incorrectly, you may want to force Suprtool to read or write one record at a time (this is what FCOPY does). To force Suprtool to read one *record* at a time, you would use:

```
>:file file.input;nomr;buf
>input file.input
>output file.output
>xeq
```

To force Suprtool to write one *record* at a time, you would use:

```
>input file.input
>:file file.output;nomr;buf
>output file.output
>xeq
```

Message Files

Message files always have variable-length records. As such, they are not accepted by Suprtool. Nor can Suprtool write them. On some releases of MPE, even opening a message file with APPEND access in Suprtool (e.g., Output msgfile, APPEND), erases the message file and may make your system run strangely.

Circular Files

Due to limitations in MPE, Suprtool can read circular files only one record at a time. Suprtool automatically detects circular input files and adjusts itself accordingly.

Suprtool cannot write out to a circular file with the append option. You will need to copy the circular file to a flat file and then append to the flat file, then write back to a new circular file.

RIO Files

Suprtool does copy and sort RIO files, but normally Suprtool reads deleted RIO records. Use a :File command to force Suprtool to ignore deleted records. The Input command prints a warning about deleted records when it opens an RIO file.

You can force Suprtool to read an RIO file as FCOPY does. If you use the following :File command on the input RIO file, all deleted records are ignored:

```
>:file inrio;buf;nomr
>input inrio
```

Suprtool must use buffered disc I/O when writing to an RIO file. This is much slower than the standard Suprtool NOBUF/MR disc I/O, but it is the only reliable way to produce an RIO file. A warning is printed in this case:

```
Warning:  Writing to RIO files is slow
```

When the output file is an RIO file, Suprtool must open the output file with NOBUF/MR access. This causes duplicate file errors if either of the following :File commands are in effect for the output file:

```
:file outfile;save
:file outfile;temp
```

You do not need to specify Save, since that is Suprtool's default. To create a temporary output RIO file, use the TEMP output option:

```
>:file outfile;rio
>output outfile,temp
```

Temporary Files

Suprtool reads and writes temporary files. No special processing must be done if the input file is temporary; just specify the file name in the Input command. By default, Suprtool saves the Output file. You must use the TEMP option of the Output command to save the file as a temporary file. The following sequence of Suprtool commands causes TEMPFILE to be left as temporary:

```
:run suprtool.pub.robelle
>in permfile
>out tempfile,temp
>exit
```

If TEMPFILE already existed as a temporary file, Suprtool would prompt you to purge the old temporary file. It you do not want to purge the old temporary file, Suprtool prompts you for a new name for the temporary file in exactly the same manner as for permanent files.

Lockwords

If you do not specify a lockword as part of the file name, MPE prompts for the lockword when Suprtool attempts to open the file. Because Suprtool must sometimes open the input or output file more than once, you may be prompted multiple times for the lockword. To avoid this prompt, we recommend that you specify any lockwords as part of the file name.

```
:run suprtool.pub.robelle
>input  infile/lockword
>output outfile/lockword
>xeq
```

This is especially important when using the erase or append option to write to a KSAM/V or KSAM/XL file with a lockword. In this case Suprtool must open the file more than once (the number of fopens vary). To avoid being prompted multiple times for the lockword, specify it as part of the output file name:

```
:run suprtool.pub.robelle
>input  infile
>output outfile/lockword.group,erase
>xeq
```

Copying User Labels

Suprtool copies all the user labels of the input file to the output file, unless you have Set Userlabels Off. The copying of user labels is done even if the output option is ,DATA or ,ASCII. If you extract fields from a self-describing (SD) file that has user labels and don't use Output,LINK or Output,QUERY, the output file still contains all the user labels of the SD file. The filecode of the output file is not SD, however.

In many instances, the user labels in a file do not interfere with the processing of the file. However, some MPE tools need specific options to work with files with user

labels. For example, FCOPY needs the Nouserlabels option when you are copying from a file with user labels to an existing file without them.

SDUnix Utility

SDUnix is an MPE program that takes self-describing file information and writes it out to an MPE flat file. This flat file can then be transferred to HP-UX together with the data file so that Suprtool/UX can reference the self-describing information about the fields.

To copy the SD file to the HP-UX machine, it must have an .sd extension and be in the same directory as the data file. For example, if the data filename is /usr/local/data/datafile, the SD file must have the name /usr/local/data /datafile.sd.

The SD file is written out to the same domain (permanent or temporary) as the input file. The SD file contains only one record with the necessary length to store all of the label information.

SDUnix Parameters

All SDUnix parameters are specified via Info = *string*. There are three parameters:

input-file sd-file LF | NOLF

The first parameter is the name of an MPE self-describing file. The second parameter is the name of the .sd file that SDUnix created.

LF vs. NOLF

Use the third parameter to specify whether the data file has LF (line feed) as the record separator, or whether the file does not use a file separator. If you use FTP to copy the data file to your HP-UX machine, you should specify the NOLF option and be sure to use a binary mode transfer. If you are using DSCOPY (with its default options) to copy the data file, you should specify the LF option.

Examples

The following section contains examples of creating an SD file on MPE, converting the SD information, and finally copying the two files to an HP-UX machine.

First create an SD file with:

```
:run suprtool.pub.robelle
>base store,5
Database password [;]?

>get d-inventory
>out dinv,link
>exit
```

Now you can convert the label information to an .sd file using the SDUnix utility. Note that the data file is the first file passed in the info string.

Specify LF if you are using DSCOPY.

```
:run sdunix.pub.robelle;info="dinv dinvsd lf"
```

Now you can use DSCOPY to copy the files to the HP-UX machine.

```
:dscopy dinv   to store.dinv   :dopey[data:password]
:dscopy dinvsd to store.dinv.sd:dopey[data:password]
```

Specify NOLF if you are using FTP.

```
:run sdunix.pub.robelle;info="dinv dinvsd nolf"
```

Use FTP to copy the files to the HP-UX machine.

```
:ftp dopey
ftp> binary
ftp> exitonerror
ftp> put dinv /users/data/store.dinv
ftp> put dinvsd /users/data/store.dinv.sd
ftp> quit
```

Now you can use Suprtool/UX to read the SD file.

```
$ suprtool
>input store.dinv
>form
   File: store.dinv     (SD Version B.00.00)  No line feeds
      Entry:                      Offset
         BIN-NO               I1      1
         LAST-SHIP-DATE       I2      3
         ON-HAND-QTY          I2      7
         PRODUCT-NO           Z8     11
         SUPPLIER-NO          Z8     19
         UNIT-COST            P8     27
         ITEM-DESC1           X20    31
              ITEM-DESC2           X20    51
         ITEM-DESC3           X20    71
         ITEM-DESC4           X20    91
   Entry Length: 110  Blocking: 1
>out dinvfile
>xeq
```

Link vs. Query

SDUnix and Suprtool/UX can recognize files created with the ,Query option and from Query. However, they cannot recognize compound item details or any Item attributes, such as Decimal or Date type.

Suprtool and Self-Describing Files

A problem with data files is that there is no field information. Self-describing files solve this problem by providing field information about the file. Suprtool reads and writes SD files; Suprlink requires SD files as input and creates an SD file as output.

Create an SD File from a Dataset

You request an SD file using the Link option of the Output command. If you extract fields from the dataset, only the extracted fields appear in the SD file.

```
>get d-sales                          {input from a dataset}
>output salefile,link                 {salefile has all of the fields from d-sales}
>xeq
```

Create an SD File from a Data File

You must Define and Extract the fields you want to have in the SD file. Use the Link option of the Output command to create the file as a self-describing file. Although

Suprtool itself allows longer field names, SD files only store the first 16 characters of a field name.

```
>input sales.data                          {input from a data file}
>define cust-no,1,6,byte
>define invoice-date,7,6,integer
>define sales-qty,13,4,packed
>extract cust-no,invoice-date,sales-qty
>output salefile,link                      {salefile has all of the extracted fields}
>xeq
```

SD Files as Input

When you specify an SD file as input to Suprtool, all the field information becomes available. You can select, extract, and total fields without the Define command.

```
>input salefile                            {self-describing file}
>form                                      {display the fields in the file}
>if sales-total > 10000                    {select based on a field}
>extract cust-account                      {only extract a few fields}
>extract sales-qty
>extract sales-tax
>extract sales-total
>total sales-total                         {total a field from the file}
>output newfile,link                       {create a new SD file}
>xeq
```

Listing SD Files

Suprtool normally lists data files in an Octal/Char format. When listing an SD file, Suprtool produces a formatted listing with field names and field values converted into ASCII:

```
>input salefile                            {self-describing file}
>list                                      {produce a formatted listing}
>xeq
```

Decimal Places and Date Formats

You use the Item command to identify items with an implied number of decimal places or a date format. If you create a self-describing file, this information is retained. When you input such a file, all Suprtool commands are automatically informed about the decimal places and date formats. The Form command shows these extra attributes as comments at the end of each field description. For example,

```
>base store.demo.robelle,5,READER
>item deliv-date ,date   ,yyyymmdd
>item purch-date ,date   ,yyyymmdd
>item sales-tax  ,decimal,2
>item sales-total,decimal,2
>get  d-sales
>output newfile,link                       {creates SD file with item attributes}
>xeq
>form   newfile                            {shows decimal pts. and dates}
```

Restrictions of SD Files

So far in this section, we have shown how to create self-describing files using the Link option of the Output command. The Link option produces a special form of self-describing file. Not all software can read this form of self-describing file. You

can use the Query option to create an old-style self-describing file. The Query option has the following restrictions.

Self-describing files were originally created by HP in MPE so that files could be fed into HPWORD and HP graphics packages. One problem with HP's definition was that no provision was made for compound fields (e.g., 10J2). When Suprtool creates an SD file with compound fields via the Query option, it uses a special data-type. When you input such a file to Suprtool, all compound fields are treated as byte arrays. Suprtool correctly copies and extracts these fields, but you cannot select with them. The Query option is not capable of retaining information about decimal places or date formats.

Creating KSAM SD Files

You can use Suprtool to create a self-describing KSAM file. There are three steps to complete:

1. Compute the number of labels (used in the KSAMUTIL or MPE/iX build command). The number is the truncated value of this calculation: `labels = ((#fields + 7) / 8) + 11`
2. Build your KSAM/V file with KSAMUTIL and specify Labels=<the number computed above>. For KSAM/XL, use the Build command with the userlabel keyword `;ULABEL=`*x* (where *x* is the number computed above).
3. Use Output xxx,Link,Erase to have the SD-information written to the file labels.

The most difficult part is computing the number of labels. You might want to use the Extract command to ensure that you know exactly how many fields will be in the output file. For example, we extract eight fields:

```
labels = ((8 + 7) / 8) + 11
       = 12
```

MPE V:

```
:run ksamutil.pub.sys
>build file2;rec=-80,16,f,ascii;keyfile=file2k; &
 key=i,6,2,,duplicate;labels=12
>exit
```

MPE/iX:

```
:build file2;rec=-80,16,f,ascii;key=(i,6,2,dup);ksamxl;ulabel=12
```

Fill the KSAM file:

```
:run suprtool.pub.robelle
>base     sales
>get      d-sales                          {input from a dataset
>extract cust-account                      {but, we extract all fields explicitly so that …}
>extract deliv-date                        {… know exactly how many fields we have}
>extract product-no
>extract product-price
>extract purch-date
>extract sales-qty
>extract sales-tax
>extract sales-total
>output  file2,erase,link                  {must have both Erase and Link}
>exit
```

Like regular SD files, when you specify a KSAM SD file as input to Suprtool, all field information becomes available. The Form command shows the structure of a KSAM SD file.

HowMessy Loadfile

HowMessy produces a self-describing file (called Loadfile) with all of the information from the HowMessy report. Suprtool is an ideal tool to process the information in this file. For example, the following Suprtool commands would select all detail datasets with a load factor greater than 85 percent:

```
:run suprtool.pub.robelle
>input   loadfile
>if      datasettype = "D" and loadfactor > 85.0
>sort    database
>sort    dataset
>dup     none,keys
>list
>xeq
```

Notes on SD Files

There are two differences between regular Suprtool output files and SD files: the filecode is not zero and there are user labels. For most applications this does not make any difference, but some applications and tools may reject these files.

When creating an SD file, Suprtool maps J-type integer items into I-type. Functionally, the two are equivalent.

Convert an SD File to a Disc File

Once Suprtool has created a file in its Output,Link format, it always treats the file as an SD file. To convert the file back to a non-SD disc file:

```
:run suprtool.pub.robelle
>in samplesd
>:file flatfile;code=0
>set userlabels off
>out flatfile
>xeq
```

Suprtool and Sorting Files

When Suprtool sorts two records that have the same key value, the first record read by Suprtool is the first record on the output file. For data files, this means that input records with the same key values appear in the same order in the output file.

Suprtool reads only the KSAM/V data file. Records in the data file are organized by chronological order. If two input records have the same key value, the one that was first added to the KSAM/V file is the first one in the output file. To help preserve sorted order in a KSAM/V file, be sure to specify all of the KSAM/V sort keys as sort keys to Suprtool.

Fast Sorting

Suprtool/iX uses a set of fast internal routines for sorting. These routines are twice as fast as the HP sort routines supplied with MPE XL 3.0, and slightly faster than the HP routines that come with MPE/iX 4.0 and 5.0.

By default, Suprtool/iX attempts to use the internal sort routines. If the internal sort initialization fails for any reason, Suprtool/iX prints a warning and tries to do the sort with HP's sort routines. If Native Language Support has been set (either automatically via the NLDATALANG JCW or explicitly via Set NLS), HP's sort routines are used.

The only way to tell which set of sort intrinsics are being used is to enable Set Stat On. The statistics from Robelle's internal sort routines are more detailed than HP's sort statistics.

Native Language Support

Suprtool determines the native language of an input source as follows:

1. The Chain and Get commands use the language of the database.
2. Using Input *filename=setname* uses the language of the database.
3. If the input file is a KSAM/XL file, the KSAM/XL language is used.
4. If the language from any of the previous steps is zero (Native-3000), Suprtool uses the language from Set NLS.

The Key and Define commands support a special *type*: CHARACTER. The CHARACTER *type* is used wherever you would use the BYTE type, but CHARACTER fields are sorted according to the native language determined by Suprtool.

To sort a French data file on the first five characters, you would use:

```
>input nlsfile                    {file with Roman-8 characters}
>set nls 7                        {sort in French}
>key 1,5,character                {new character data-type}
>output sortfile
>xeq
```

With Reduced Disc Space

If your system is almost out of disc space, you may find that Suprtool operations are being stopped because the "Sort Library" cannot build its scratch file, Sortscr. A common method of overcoming this error is to use the Numrecs command.

By default, Suprtool assumes that the OUTPUT file and the Sortscr file must be able to contain *every* input record. Even though you are reading a dataset of one million records, you may *know* that only 1,000 records will be selected. If you do *not* use the Numrecs command, Suprtool builds an output file with one million records and the sort builds a scratch file with one million records.

If your dataset was called ARTRANS, the following Suprtool commands would be used to read, sort, and select a small subset of the input records, preventing the out-of-disc space error:

```
:run suprtool.pub.robelle
>base actrec.data
>get artrans
>numrecs 1000
>if trans-date>=000101 and trans-date<=000131
>sort trans-date ;sort cust-number
>output archives
>exit
```

Suprtool and Tape Files

You must use a :File command to specify an input or output file to tape. Suprtool reads or writes only *one* file to an unlabeled tape. Suprtool writes two end-of-file marks on an Output tape to guarantee that any other software can find the end-of-file correctly.

If you plan to select more than 10,000 records from a tape file, be certain to specify the maximum number of records to select in the Numrecs command. The Numrecs command is not necessary if you are outputting to an existing file with the append and erase options of the output command.

There is an outstanding bug in MPE that can cause Suprtool to lose records from an Input tape file. This only happens where there are partial blocks on the tape that are *not* the last block of the tape. Suprtool cannot detect this situation, but the workaround is to specify NOMR on the input :File command for the tape. For example:

```
:file intape;dev=tape;rec=-80,50,f,ascii;nomr
```

Another recurring problem with MPE and tape files is odd-byte length records. We suggest that when writing to tape, even-byte length records be used. These records take exactly the *same* amount of space on tape or disc, because MPE always rounds up the record length to the nearest full word boundary. To read data from an IBM tape where records start on odd-byte offsets, use the DEBLOCK option of FCOPY.

Suprtool cannot read or write EBCDIC tapes; use FCOPY instead.

Labelled Tapes

Specifying DEV=TAPE on a :File command causes MPE and Suprtool to create a tape that has one file. If you need to have multiple files on one tape or if your file takes more than one reel, you must use labeled tapes.

To use a labeled tape requires a different kind of :File command. A typical :File command might be:

```
:file o;dev=tape;label="VOL1",,,next
```

The string "VOL1" is the name of this volume of tapes. This name appears on the console. Note that it is possible to specify volume names in lowercase letters. Never do this, as it becomes extremely confusing when working with the console.

Use the "NEXT" keyword when writing the first file to the tape. Be careful with the "NEXT" keyword, because you can overwrite an existing labeled tape, even though it has not expired.

Use the "ADDF" keyword to add more files to the tape. This tells MPE to add each file to the end of the tape volume. Use the file name "*O" with the Suprtool Output command.

You do not need to specify a record size or blocking factor on the File command. When the file is read back, most of the file attributes will be remembered. One exception to this is file code. Files read from a labeled tape always have a file code of 0, unless you use a :File command to specify a specific file code.

Multiple Files

Do the following to store four files to a labeled tape:

```
:run suprtool.pub.robelle
>:file o;dev=tape;label="VOL1",,,next
>in a;out *o;x                          {first file on the volume}
>:file o;dev=tape;label="VOL1",,,addf
>in b;out *o;x                          {subsequent files are added to the end}
>in c;out *o;x
>in d;out *o;x
```

If all of these files are purged and you want to restore the file "D", you would do the following:

```
:file i;dev=tape;label="VOL1",,,4
:run suprtool.pub.robelle
>in *i;out d;x
```

The "4" on the :File command instructs the file system to restore the fourth file on the tape. If you forget the contents of a labeled tape, make FCOPY read it as if it were unlabeled:

```
:file x;dev=tape;acc=update
:fcopy
from=*x;to=;char                        {displays the volume and the first file headers}
from=*x;to=;char;skipeof=+3             {displays the file header for the following file}
```

You will be asked for permission to scratch the label, but MPE does not scratch the label unless you write to the tape.

Notes

There are two things to remember when working with multiple tapes in a single volume set. The first tape reel in a volume has the same name as the entire volume set. Subsequent tapes should have a different name. If you know that the file you want to read is on the second reel, you can just mount the second reel.

File User Labels

When copying a file to a labeled tape, Suprtool attempts to copy any MPE user labels. Because of restrictions of labeled tapes, the user labels cannot be longer than the record length of the file. In most cases, Suprtool truncates user labels as they are copied to or from a labeled tape. When this happens Suprtool prints a warning:

```
Label length truncated to 80 characters
```

If the files you are reading from the tape do not have any user labels, and Suprtool is having difficulty reading the file, you can try this:

```
>Set Userlabels Off
```

Suprtool and Remote File Access

The Base command allows you to specify a *system* name, other than the logon system. This permits easy access to databases on remote computers. :File commands can be used to specify input files from remote computers and to specify output files to remote computers.

Suprtool attempts to handle duplicate file errors on a remote computer in the same manner as for a local computer. If you are on-line, you can rename the file as a file on the remote computer. In batch, a duplicate file on a remote computer results in the Outputnn file being built on the remote computer.

Due to bugs in MPE, Suprtool cannot always rename files on a remote computer. We suggest that output files on remote computers be purged before you enter the Exit or Xeq command.

Network Services (NS)

NS/3000 (Network Services for the HP e3000) is the replacement for DS over LAN/3000. Suprtool reads and writes files using NS/3000. The Base, Input, and Output commands do not allow the *node specification* to be entered as part of the file name. Use File commands to specify remote databases and files. For example,

```
:dsline mrp.go.hp
:remote hello user.account,group
:file store=store:mrp.go.hp
:run suprtool.pub.robelle
>base store
>get m-customer
>output mcust
>xeq
```

See "Base Command [BA]" on page 132 for a discussion of remote database access.

Suprtool and MPE/iX

Suprtool is available in native mode. This section provides details for calling the Suprtool2 interface in native mode. See the next section for ideas on how to combine MPE/iX programming features with Suprtool.

Calling Suprtool in Native Mode

The Suprtool2 interface is available in native mode. Like the compatibility mode version of the Suprtool2 interface, you require PH capability in order to use the Suprtool2 interface in native mode. The native-mode object code for the interface resides in ST2XL.Pub.Robelle.

The interface is installed in XL files. The following commands demonstrate how to compile and link your program to use the native-mode Suprtool interface.

```
:cob85xl example.source
:link from=$oldpass;to=example.pub;cap=ph
:run example.pub;xl='st2xl.pub.robelle'
```

User XL Files

You can add the native-mode Suprtool2 interface to your own XL file. For example,

```
:linkedit
-xl xl.pub
-copyxl from=st2xl.pub.robelle;module=suprtool.asm
-exit
:run example.pub;xl='xl.pub'
```

When Errors Occur

On MPE V, if you have not installed the Suprtool2 interface, you see an error message such as:

```
:run testprog
UNRESOLVED PROGEXTERNAL SUPRTOOL2
UNABLE TO LOAD PROGRAM TO BE RUN.  (CIERR 625)
```

If you forget to specify the XL file on MPE/iX, you get a similar error message.

Sorting with Suprtool/iX

The native mode HP sort routines do not sort files with records greater than 4096 bytes. Suprtool/iX uses these native mode HP sort routines with NLS or with Set Sortfast Off. In all other cases, Suprtool/iX uses its own set of sort routines that have no limit on the record size.

If you attempt to sort records greater than 4096 bytes with Suprtool/iX calling the HP Sort, you get the following error message:

```
Error:  Sort record size is greater than 4096 bytes
The native mode HP Sort intrinsics are not able to sort
records greater than 4096 bytes in length.  If you want
to sort records greater than 4096 bytes, use the
compatibility mode version of Suprtool.  To improve speed
we recommend using the Object Code Translator on the CM
version of Suprtool:

    :hello mgr.robelle,pub
    :octcomp suprcm,suproct
    :run suproct.pub.robelle
```

We consider this problem a bug in HP's native mode sort intrinsics, one unlikely to be fixed. The solution is to use the Object Code Translated version of Suprtool to sort files with records greater than 4096 bytes.

Suprtool and CI Variable Substitution

Suprtool is able to substitute any CI (command interpreter) variables from almost any command line source, whether via interactive commands, a usefile, or batch input. In order to use this feature, first enter the following Set command:

```
set varsub on
```

Variable substitution is not enabled, by default, for backward compatibility.

Example

Because Suprtool is now capable of CI variable substitution, you no longer need to echo commands out to a file and subsequently use the newly created file.

The following example varies the sort order of an extract using a CI variable called "order."

```
!setvar order 'desc'
!run suprtool.pub.robelle
set varsub on
base test
get dline
sort ord-num,!order
output file3y
exit
```

Batch Requires Indent

The Streams facility (under default setup) replaces any exclamation mark (!) found in the first column of a job stream. Anytime you want to specify an entire line through variable substitution, you need to leave a space before the variable is specified.

```
!setvar a 'base mydb;get orders;output file3x;exit'
!run suprtool.pub.robelle
set varsub on
 !a                                          {indent this line}
```

Avoiding Double Resolution

For MPE commands, some variables are resolved twice when passed off to MPE. A double resolution gives different values if a variable references other variables.

The following example sets variables at the CI level:

```
MPEXL:setvar a 10
MPEXL:setvar b "!!a"
MPEXL:showvar [ab]
A = 10
B = !a
```

Setting the variables within Suprtool with variable substitution enabled, changes the value of b.

```
>{different results from MPE CI}
>set varsub on
>setvar a 10
>setvar b "!!a"
>showvar [ab]
A = 10
B = 10
```

Suprtool does one level of variable substitution before the command is passed off to MPE. If you are setting variables that reference other variables, we recommend you set them before running Suprtool or temporarily disable variable substitution with the Set Varsub Off command.

```
>{same results as MPE CI}
>set varsub off
>setvar a 10
>setvar b "!!a"
>showvar [ab]
A = 10
B = !a
>set varsub on
```

Suprlink and STExport

Suprtool, Suprlink, and STExport each have their own Set Varsub command. To do variable substitution in any of these components, you have to use the Set Varsub On command in the appropriate component.

Because Suprtool automatically looks in the Suprmgr.Pub.Sys file for any configuration commands, that is a good place to put the Set Varsub On command. Suprlink and STExport, however, do not use Suprmgr.Pub.Sys and do not have their own configuration files in which to look. If you run Suprlink and STExport directly from the command line, you must enable variable substitution explicitly:

```
!run suprlink.pub.robelle
set varsub on
```

One way to ensure that Suprlink and STExport automatically get variables substituted is to invoke Suprlink and STExport from the Suprtool command line. In this way, variable substitution is actually done by the Suprtool command parser before it passes the commands to Suprlink and STExport.

For example, suppose the title of the HTML report is stored in a variable named "title." Pass commands to STExport using Suprtool's Export *exportcommand* syntax:

```
!setvar title 'Delinquent Accounts'
!run suprtool.pub.robelle
set varsub on
{execute task to create self-describing file foo}
export input  foo
export html   table title '!title'
export output ./delinq.html
export exit
exit
```

Note that the following sequence of commands would **not** have worked, where STExport is invoked from Suprtool using the Export command with no parameters. Variable substitution would not take place in this case:

```
!setvar title 'Delinquent Accounts'
!run suprtool.pub.robelle
set varsub on
{execute task to create self-describing file foo}
export
input  foo
html   table title '!title'
output ./delinq.html
exit
exit
```

Suprtool and User Prompting

Suprtool does not contain any features for prompting of input values, but it does have support for MPE/iX variables. Instead of input values, you can use MPE/iX to create a file of Suprtool commands and run Suprtool with this file as input.

Example

Suppose you have a Suprtool usefile called Suprcmd. Use, which contains the following:

```
set varsub on
base store,5,reader
get m-customer
:input credit,"Enter Credit Limit :"
if credit-rating >= !credit
list stan
exit
```

You can then invoke the command file and subsequent prompting from within Suprtool with the following command:

```
:run suprtool.pub.robelle;parm=4;info="use suprcmd.use"
```

You can also prompt for variables using the MPE Input command in any command file and reference them as long as variable substitution is enabled with the command:

```
set varsub on
```

Resolving Variables

Suprtool, now has a setting called Set VarsubDebug on which will print out the line after the variable substitution has occurred. This setting only works if Set Varsub is on and Set VarsubDebug is on.

```
setvar outfile &
:"/GREEN/SUPRTEST/filename90123456789012345678901234567890123456789012345678901"
:run suprtool.pub.robelle
SUPRTOOL/iX/Copyright Robelle Solutions Technology Inc. 1981-2007.
(Version 6.2 Internal)  TUE, OCT 30, 2007,  2:58 PM  Type H for he
>set varsub on
>set varsubdebug on
>in file1sd.suprtest
vd:in file1sd.suprtest
>output !outfile,link,temp
vd:output /GREEN/SUPRTEST/filename90123456789012345678901234567890123
vd:2345678901,link,temp
```

The output is formatted into 74-byte chunks and printed with a preceding “vd:” so the “substituted” line is clear.

Suprtool and Personal Computers

You can format files to be downloaded to your PC for use in spreadsheets or databases with the PRN option of the Output command. Suprtool formats your file as a PC structure (a comma-delimited file). Not all PC applications support the PRN format. For more precise data conversion, create a self-describing file then use STExport. See the STExport manual for details. You transfer the Suprtool output file to your PC and then import it into your PC application.

Downloading to the PC

After you have created a PRN file using Suprtool, you can use FTP or any of the many terminal emulator programs available to download the file to the PC. This includes Reflection from Walker Richer & Quinn. Robelle's text editor, Qedit, is able to execute Reflection commands from the HP e3000, helping to automate the process of downloading.

Decimal Places

Be sure to specify which fields have decimal places when creating the PRN file. Suprtool reserves extra space for decimal points that appear in the PRN output. When formatting numeric fields, Suprtool inserts the decimal point at the correct place. When you import your file into your PC application, numeric fields are automatically formatted correctly.

Spreadsheets

The following procedure allows you to include literal headings in your spreadsheet using only one file, the size and shape of which is computed by STExport. We have tested this method with MS Excel; it should work with any spreadsheet that supports the importing of delimited files.

There are two steps. First, build a self-describing file with Suprtool, then use STExport to convert it to PRN and add the headings.

1. In Suprtool build a self-describing file:

```
>input   ...
>define  items...
>item    items...
>extract fields...
>output  sdfile,link
>xeq
```

2. In STExport convert to PRN and add the header line:

```
$input   sdfile
$heading fieldnames
$output  pcdata
$exit
```

The file Pcdata is a variable-length PRN file with both headings and data.

Suprtool and PowerHouse Applications

You can use Suprtool to significantly speed serial extracts using Quiz and QTP from Cognos. In many cases the changes to existing applications are minor.

Suprtool with Quiz/QTP

Quiz and QTP are part of PowerHouse, a popular fourth-generation language sold by Cognos. You can improve the performance of Quiz reports by using Suprtool to select and sort the records from a dataset, then pass the selected records to Quiz for final reporting. To do this you need some way to tell Quiz about the record structure of Suprtool's output file. Quiz already has the capacity to do this without making any changes to the PowerHouse dictionary (QSCHEMA).

In the following example, Suprtool extracts entries from the Custmast dataset, sorts them, and writes them to the Cmasfile MPE file. These are the records we need for the Quiz report.

Step 1: Create a Subfile with Quiz

The first step is to use Quiz to extract one entry from the Custmast dataset and write it to a PowerHouse "subfile". Note that you could also use QTP to build the subfile. This is described shortly.

```
:purge cmasfile
:quiz
>access custmast
>set report limit 1
>set subfile name cmasfile keep size numrecs
>report summary all
>go
```

Replace the Numrecs parameter with the number of records that will be extracted by the Suprtool run. The default Numrecs is 1 when the report limit is set to 1.

Remove the Keep parameter to create a temporary file instead of a permanent one. Suprtool can write to temporary files as easily as to permanent files. Temporary files have the advantage that when multiple batch jobs are running at the same time and trying to use the same file names, the files won't interfere with each other. Permanent files, on the other hand, may not have to be rebuilt every time. The same permanent subfile can be used repeatedly, as long as the record structure doesn't change. The entire first step of running Quiz or QTP to create the subfile may be omitted.

The Set Subfile statement in Quiz creates a PowerHouse subfile. Subfiles are regular MPE files (or KSAM/XL files, in the case of indexed subfiles) which have user labels containing their record descriptions, in Cognos' proprietary format. The data portion of the files is standard, however, and can be read by other (non-Cognos) products. It is important to ensure that when you read a subfile with Suprtool, you define the fields correctly so that Suprtool's view of the data structure exactly matches the PowerHouse description of the records.

Step 2: Output Erase in Suprtool

Once you have created the PowerHouse subfile, use the Erase option of the Output command in Suprtool to load the file. This overwrites any data in the subfile, but it does not touch the PowerHouse mini-dictionary in the user labels:

```
:run suprtool.pub.robelle
>base actrec
>get custmast;if cred-limit>=1000000
>output cmasfile,erase
>sort custnum
>exit
```

Step 3: Report with Quiz

The Cmasfile file contains the sorted records for the Quiz report. Quiz knows the structure of this file because of the initial Quiz commands that we used to create it. You can now use Quiz to generate the report:

```
:quiz
>access *cmasfile
>report ...
>go
```

Using QTP to Create Subfiles

You can also use QTP instead of Quiz to create the PowerHouse subfile:

```
:purge cmasfile
:qtp
>access custmast
>set input limit 0
>subfile cmasfile size numrecs keep include custmast
>go
```

There are a number of reasons to use QTP instead of Quiz to create subfiles:

Quiz subfiles always reflect a record structure based on the lowest level of redefinition in the dictionary. So, if an item is redefined, it appears once for each redefinition in the dictionary. For example:

```
Record CUSTOMERS
  Item CUSTOMER-NUM               character size 6
  Item CUSTOMER-NAME              character size 40
  redefined by
     Item COMPANY-NAME            character size 40
  end
  redefined by
     Item FIRST-NAME              character size 20
     Item SURNAME                 character size 20
  end
```

The customer-name field is redefined twice in the dictionary. Although the record size of the file is only 46 bytes, a Quiz subfile created with

```
>report summary all
```

would create a subfile with a record length of 86 bytes as follows:

```
Item CUSTOMER-NUM               character size 6
Item COMPANY-NAME               character size 40
Item FIRST-NAME                 character size 20
Item SURNAME                    character size 20
```

A subfile created with QTP, on the other hand, would exactly match the record structure of the source file. Therefore, it is always safer to use QTP to create the subfile, as it is more likely to match the structure of Suprtool's output record.

With QTP's Include parameter, it is easy to include either specific fields from the record or all of the fields. You can also easily include fields from other datasets. This is handy when creating a subfile that holds the output from a Suprlink task.

Creating Subfile with Command File

The process of creating a subfile is essentially the same regardless of the database and dataset being used. The only things that change are the subfile and dataset names. The task can easily be automated using the MPE/iX Makesub command file. This command file can be executed before running Suprtool, or from within Suprtool.

Here is the simplest form of the Makesub command file:

```
parm subfile, dataset
echo set report limit 1                          >usefile
echo access !dataset                             >>usefile
echo set subfile name !subfile size 10000        >>usefile
echo report summary all                          >>usefile
echo go                                          >>usefile
echo exit                                        >>usefile
run quiz.current.cognos; &
    info="auto=usefile nolist" >$null
```

Makesub can be invoked from within Suprtool, as shown below:

```
:run suprtool.pub.robelle
>base actrec
>:makesub cmasfile, d-custmast
>get custmast;if cred-limit>=1000000
>output cmasfile,erase
>sort custnum
>exit
```

The subfile is created from the single statement :makesub from within the Suprtool task.

The preceding form of the Makesub command file achieves the goal of simplifying the Suprtool code, but it can still be improved. A better form would incorporate a few more features for added flexibility. The following version of Makesub provides these advantages:

1. The capacity of the subfile can be specified.
2. The subfile can be temporary or permanent.
3. The subfile can contain either all the elements of a file, or selected elements.

QTP is used instead of Quiz so that there are no multiple redefinitions of any item. This helps ensure that the layout of the subfile matches the output record that Suprtool creates.

```
parm subfile, dataset, temp_or_not=" ", flimit=10000
anyparm include=!dataset

if "!temp_or_not" = "TEMP"
   setvar keep " "
   setvar purgetemp ",temp"
else
   setvar keep "keep"
   setvar purgetemp " "
endif

file s=!subfile,old!temp_or_not

if finfo("*s",0)
   purge !subfile !purgetemp
   echo Existing subfile has been purged
endif

echo set input limit 0                           >usefile
echo access !dataset                             >>usefile
echo subfile !subfile size !flimit !keep &
     include !include                            >>usefile
echo go                                          >>usefile
echo exit                                        >>usefile
run qtp.current.cognos;info="auto=usefile nolist">$null
purge usefile,temp
echo Subfile has been created
```

The new-and-improved version of Makesub can be invoked from within Suprtool, as shown below:

```
:run suprtool.pub.robelle
>base actrec
>:makesub cmasfile, d-custmast, TEMP, 5000, &
     customer-number, company-name, first-name
>get custmast;if cred-limit>=1000000
>extract customer-number, company-name, first-name
>output cmasfile,erase
>sort custnum
>exit
```

Suprtool and PowerHouse Data-Types

One of the recurring themes in the Technical Support calls that we receive involves identifying data-types when defining fields in Suprtool. This is particularly true of PowerHouse sites, because PowerHouse views the data through its dictionary definitions, while Suprtool's vocabulary is based on IMAGE terminology. Because both products read the same underlying data structures, the problem is only one of semantics. This section aims to assist the user in correctly declaring data structures that are passed between Suprtool and PowerHouse processes.

Translation Table of Data-Types

IMAGE Type	PowerHouse Type	Suprtool Type
U or X	Character	Byte (Character for NLS)
R	Float (non-IEEE)*	Real or Long
E	Float (IEEE)*	IEEE
K	Integer Unsigned	Logical
J	Integer Signed	Integer or Double
P	Packed	Packed
Z	Zoned	Display

(* IEEE or Non-IEEE depends on whether "float IEEE" or "float NONIEEE" was declared under System Options in your system dictionary)

Tip 1: Use QSHOW to examine PowerHouse data structures.

Many users are familiar with Quiz's Show Items command:

```
> access d-sales
> show items
                          INPUT OUTPUT
 D-SALES                 TYPE SCALE SCALE  DEC PICTURE
 * CUST-ACCOUNT          NUM     0     0    0  ^^^^^^^^
   DELIV-DATE            NUM     0     0    0  ^^^^^^^^^^
 * PRODUCT-NO            NUM     0     0    0  ^^^^^^^^
   PRODUCT-PRICE         NUM     0     0    0  ^^^^^^^^^^
   PURCH-DATE            NUM     0     0    0  ^^^^^^^^^^
   SALES-QTY             NUM     0     0    0  ^^^^^
   SALES-TAX             NUM     0     0    0  ^^^^^^^^^^
   SALES-TOTAL           NUM     0     0    0  ^^^^^^^^^^
```

This listing shows you the attributes and sequence of the fields but doesn't give any clues as to their storage format.

QSHOW gives a much more useful listing:

```
> show subfile salesub
                SUBFILE DICTIONARY              Page    1
                SUBFILE RECORD REPORT

   Subfile:            SALESUB
   Type:               Permanent
   Organization:       Direct
   Format:             7
   Record Size:        38 Bytes

 Item                  Type            Offset Size  Occurs

 CUST-ACCOUNT          ZONED UNSIGNED   0      8
 DELIV-DATE            INTEGER SIGNED   8      4
 PRODUCT-NO            ZONED UNSIGNED   12     8
 PRODUCT-PRICE         INTEGER SIGNED   20     4
 PURCH-DATE            INTEGER SIGNED   24     4
 SALES-QTY             INTEGER SIGNED   28     2
 SALES-TAX             INTEGER SIGNED   30     4
 SALES-TOTAL           INTEGER SIGNED   34     4
```

Notice that this listing shows the offset of the fields within the record (with the first field starting at offset 0), the length (size) of the fields, as well as the storage data-type. In most cases, these map directly onto the IMAGE/Suprtool data-types. (See Translation Table above.)

For example, the above subfile can be defined in Suprtool as follows:

```
>define cust-account,1,8,display
>define deliv-date,9,4,integer
>define product-no,13,8,display
>define product-price,21,4,integer
>define purch-date,25,4,integer
>define sales-qty,29,2,integer
>define sales-tax,31,4,integer
>define sales-total,35,4,integer
```

Note that PowerHouse's Zoned Unsigned data-type is equivalent to Suprtool's Display data-type, and that Suprtool's Define offset starts at 1, which is one higher than the offset in the QSHOW listing.

Tip 2: Where is the Data Coming From?

When a PowerHouse subfile contains complete records from an IMAGE dataset, you can have Suprtool read the record structure directly from the IMAGE Root file:

```
>base store,1,WRITER
>input salesub = d-sales
>form

Database: STORE.DEMO.ROBELLE

    D-SALES          Detail                  Set 5
        Entry:                       Offset
          CUST-ACCOUNT          Z8       1  (!M-CUSTOMER)
          DELIV-DATE            J2       9
          PRODUCT-NO            Z8      13  (M-PRODUCT)
          PRODUCT-PRICE         J2      21
          PURCH-DATE            J2      25
          SALES-QTY             J1      29
          SALES-TAX             J2      31
          SALES-TOTAL           J2      35
    Capacity: 602 (14)  Entries: 8  Highwater: 8  Bytes: 38
```

Of course, if the subfile contains only some of the fields, then you'll need to define them individually. But you can still use Suprtool's Form command to examine their storage structure in the datasets from which they've been extracted.

Tip 3: Defined Fields' Data-Types

In some cases, subfiles may contain fields that were created within a previous Quiz task, namely:

- Defined fields
- Results of summary functions (Count, Subtotal, etc ...)

Defined fields will have whatever storage type was specified when they were created. If storage type "numeric" was specified, PowerHouse will create a Float Size 8 field by default. It is also important to note that if the Define command is not explicit about the field size, PowerHouse will assume a size based on the requested number of digits. For example,

```
define D-PRICE integer*4 = 25
```

creates a field capable of storing 4 digits (i.e., a 2-byte integer), while

```
define D-PRICE integer size4 = 25
```

will guarantee a 4-byte integer field. When in doubt, use QSHOW to examine the structure of the subfile.

Tip 4: Fields Created by Summary Operations

Here is a list of data-types for PowerHouse summary functions written to subfiles:

Data-Type	Field
Count	Integer size 4
Subtotal	Float Size 8
Minimum	Float Size 8 for numerics, Character size nn for char fields
Maximum	Float Size 8 for numerics, Character size nn for char fields
Average	Float size 8 subtotal, Integer size 4 count, and an Integer size 4 filler of binary zeros.
Percent	two Float Size 8 subtotals
Ratio	two Float Size 8 subtotals
Sysname	Character size 40
Sysdate	Phdate integer size 2 or 4 (if century included)
Systime	Integer size 2 (HHMM format)

Suprtool Definitions - QSCHEMA

You can create Suprtool field definitions from the record layout information provided by PowerHouse. Think of the QSHOW output as a list of records that needs only to be translated into another form. What you do is send the QSHOW report to a disc file, and process that file with Quiz, turning it into a list of Define commands for Suprtool.

Notes on QSHOW Output

To capture the output of QSHOW, you can run QSHOW and redirect the $stdlist. Or, you can set a :File command for QSHOLIST to a disc file:

```
:build templist;rec=-80,,f,ascii
:file qsholist=templist,old;dev=disc
:run qshow.current.cognos
```

Once inside QSHOW, you must Set Report Device Printer so that QSHOW uses the QSHOLIST file. The advantage of using QSHOLIST is that only the actual file report is put in the file, not any of the other information printed by QSHOW (such as the program banner).

PowerHouse subfiles can be processed in the same way as files defined in the PowerHouse dictionary (QSCHEMA). There is a QSHOW command called Show Subfile, which works identically to Show File and produces a report with the same information. Some of the columns are off horizontally, but our Quiz example program fixes that.

Use the QSHOW command Set Compressed to suppress the page numbers and other heading information in the QSHOW report.

A sample output from QSHOW would look like this:

```
    Subfile:        F184A.PC (Permanent)
    Format:         3
    Record Size:    138 Bytes
Item        Type            Offset    Size      Occurs
PRONUM      CHARACTER       0         18
INSHP       FLOAT           18        4
REFNUM      CHARACTER       22        10
SONUM       CHARACTER       32        10
ARFSC       CHARACTER       42        4
TYPNUM      INTEGER         46        2
SOES1       CHARACTER       48        30
SOES5       CHARACTER       78        30
BILNAM      CHARACTER       108       30
```

First, we define what the QSHOW reports look like in a dictionary:

```
File Qshow-Subfile
     Organization Direct &
     Open QSHOLIST
File Qshow-File
     Organization Direct &
     Open QSHOLIST

Record Qshow-Subfile
  Item Filler1                    character size 1
  Item Name                       character size 23
  Item Type                       character size 18
  Item Offset                     freeform  size 10
  Item ItemSize                   freeform  size 10
  Item Occurs                     freeform  size 18

Record Qshow-File
  Item Key                        character size 6
  Item Name                       character size 35
  Item Type                       character size 18
  Item Offset                     freeform  size 4
  Item ItemSize                   freeform  size 5
  Item Occurs                     freeform  size 12
```

Note that we have two definitions, one for the Show Subfile output and one for the Show File output. We use the same names for the columns and adjust the Size to accommodate the differences in horizontal positioning. Note that Freeform is a number stored in a character field; Quiz automatically converts it to a number.

The Quiz Report

It is not essential to use Quiz to do the definition translation; a COBOL program could be used. The object of the Quiz report is to produce a Suprtool Define command for each field. The Define command has the following components:

1. The word "define".
2. The field name.
3. The position (starting with column 1). The QSHOW report gives an offset starting at zero, so the position is the offset plus 1.
4. The field size in bytes.
5. The field type, using the Suprtool names.
6. The subcount (Quiz's occurs column).

We have one Suprtool Define command for each line of the QSHOW report.

Remember that we mean to define just the lines that represent field information. Using the above example of QSHOW output, we do not want to process the first six lines, because they are heading information. For these lines, however, the column containing the field size does not contain a valid number. We can simply tell Quiz to select only those records where ItemSize is greater than zero.

We get the field name from the name already provided in the data dictionary. To get the field type, we can create a Quiz Define statement which maps the PowerHouse types into the Suprtool types. Although the PowerHouse type sometimes has a sign qualifier for numeric fields, in this case it doesn't matter. We need to check only the first few characters of the type. We convert a blank in the occurs-column to a subcount of one.

```
;QUIZ - SUPRTOOL Definitions
Access QSHOW-SUBFILE         ;or QSHOW-FILE
Select if ItemSize > 0       ;Ignore heading lines
Define Position = Offset + 1
Define SuprType char*8 = &
  "integer" if Type[1:8]="INTEGER" else &
  "packed"  if Type[1:7]="PACKED" else &
  "real"    if Type[1:6]="FLOAT" else &
  "display" if Type[1:6]="ZONED" else &
  "byte"
Define SuprSubcount char*8 = &
  "1" if Occurs = 0 else &
  ascii(Occurs)
Report &
  tab 1 "DEFINE" Name pic"^^^^^^^^^^^^^^^^" &
  ","  Position pic "^^^" &
  ","  ItemSize pic "^^^"  &
  ","  SuprType pic "^^^^^^^" &
  ","  SuprSubcount
set nohead
set page length 1  report spacing 1
set report device disc name SUPRDEFS   ;send to disk file
go
```

The output from the above example would look like this:

```
DEFINE PRONUM                ,   1 ,   18 , byte   , 1
DEFINE INSHP                 ,  19 ,    4 , real   , 1
DEFINE REFNUM                ,  23 ,   10 , byte   , 1
DEFINE SONUM                 ,  33 ,   10 , byte   , 1
DEFINE ARFSC                 ,  43 ,    4 , byte   , 1
DEFINE TYPNUM                ,  47 ,    2 , integer, 1
DEFINE SOES1                 ,  49 ,   30 , byte   , 1
DEFINE SOES5                 ,  79 ,   30 , byte   , 1
DEFINE BILNAM                , 109 ,   30 , byte   , 1
```

This represents a simple use of definition processing from the data dictionary. There are many other uses for this technique. These are some other examples: to create record definitions for FORTRAN programs (define variables for each field, and EQUIVALENCE them to the proper element of a buffer array), for tape file transfers to other machines (create a new file with all numeric fields converted to type Zoned), and to create PC database structure definitions for receiving downloaded data (also see "Format Options" on page 235).

Quiz: Generating Suprtool Commands

Suprtool does not have the ability to prompt users for selection criteria. You can easily create a short Quiz procedure to prompt for values and create a file of Suprtool commands.

```
> define D-PROMPT date format YYMMDD =            &
>    parm prompt "Enter selection date: "
> report &
>    "base BASENAME,5"        &
>    skip "get SETNAME"       &
>    skip "if DATEFIELD = " D-PROMPT pic "^^^^^^" &
>    skip "output OUTFILE"    &
>    skip "xeq"
> set nohead
> set report device disc name STCODE
> set page length 0
> go
```

Quiz edits the user's input for a valid date. Note that there is no ACCESS statement, but Quiz still writes one record to the output file. Execute the resulting commands as a Suprtool usefile:

```
:run suprtool.pub.robelle;parm=4;info="use stcode"
```

Generating Suprtool Commands from Quick

Alternatively, you could use Quick to prompt a user for input, to create a file of suprtool commands:

```
screen test menu temp start-date char*8 temp end-date char*8 temp
prompt char*40 define D-CMD char*80 = "ECHO if Start-date !> " + start-
date &
  + " and end-date !< " + end-date + " >> mytemp" title "Creating Use
File Demo Screen" centered skip 2 field prompt field start-date field
end-date procedure initialize begin
  let prompt = "Enter a start-date (YYYYMMDD)"
  display prompt
  accept start-date
  let prompt = "Enter an end-date (YYYYMMDD)"
  display prompt
  accept end-date
  run command "echo base STORE.DEMO.ROBELLE,1,WRITER > mytemp"
  run command "echo get D-SALES >> mytemp"
  run command D-CMD
  run command "echo output OUTFILE,link >> mytemp"
  run command "echo exit >> mytemp"
  run command "run SUPRTOOL.PUB.ROBELLE;parm=40;info='use mytemp'" end
build
```

This doesn't require any special dictionary declarations, as it uses MPE "echo" commands to create the file.

Suprtool and Application Systems

Suprtool can be used with fourth-generation languages and other tools.

Third-Party Indexing

Omnidex is a product from Dynamic Information Systems Corporation (DISC). Superdex is a product from Bradmark Technologies, Inc. Each product provides generic and keyword search capabilities for IMAGE data. It is necessary to keep their index information up-to-date when any program, including Suprtool, adds or deletes records from the indexed database.

In MPE/iX 4.0, Hewlett Packard has provided an interface that ensures that the Omnidex and Superdex indexes remain up-to-date. You only need to enable third-party indexing (TPI) using DBUTIL. Once this is enabled, MPE automatically calls the TPI intrinsics to maintain the indexes as records are added, delete and modified.

```
:run dbutil.pub.sys
>>enable mybase for indexing
>>exit
```

It is important to note that Suprtool accesses third-party indexes by the Chain command, which in turn invokes DBFIND mode-1 to establish the chain and DBGET mode-5 to read down the chain. It is important that you follow whatever rules are required by your indexing product to use its respective third-party index, as opposed to any IMAGE index that may exist.

For example, with Omnidex you might need to set a JCW used by Omnidex called IMAGETPI to determine which index or chain is used.

For more detailed information, consult the *TurboIMAGE Reference Manual*, and the appropriate documentation from DISC or Bradmark.

Z-type TPI-keys

Suprtool supports stand-alone “Z” type keys. Suprtools form command will show the keys that it can support via the chain command.

The length of a key is returned by dbinfo-833 and Suprtool will report in the form command the length of a stand-alone key. (SI-PRODUCT-NO is the path that is now supported) Superdex has a variable that impacts the length that is returned for a given key, and that is SICOGNOS. Suprtool will account for the length difference on the chain command but will show the unadjusted length. Normally Superdexs dbinfo-833 would show a length of 12 for a Zoned-decimal tpi-key with a length of 8, if SICOGNOS is set to 1, then Superdex will return 8.

```
form d-inventory

Database: STORE.SUPRTPIS.GREEN  TPI: SUPERDEX (010566) 4.2.11.2

    D-INVENTORY      Detail                   Set# 4
       Entry:                       Offset
          BIN-NO                 J1       1
          LAST-SHIP-DATE         J2       3
          ON-HAND-QTY            J2       7
          PRODUCT-NO             Z8      11  (M-PRODUCT)
                                             <<TPI>>
          SUPPLIER-NO            Z8      19  (!M-SUPPLIER)
          UNIT-COST              P8      27
          ITEM-DESC1             X20     31  <<TPI>>
          ITEM-DESC2             X20     51
          ITEM-DESC3             X20     71
          ITEM-DESC4             X20     91
    Capacity: 464 (8)  Entries: 13  Highwater: 13  Bytes: 110

    Additional Third-Party Indexes:
          SI-PRODUCT-NO          Z8      B
          SI-ITEM-DESC1          X20     B
          SI-ITEM-DESC1234       X80     B
          SHAD                   X10     B
```

Omnidex without TPI

On MPE V, and on versions on MPE/iX that do not support TPI, you must access the Omnidex intrinsics via "call conversion". To use call conversion with Omnidex version greater than 2.05.03 and less than 3.00, Suprtool.Pub.Robelle must be copied to the CC group of the DISC account and this copy of Suprtool must be run.

```
:run suprtool.cc.disc;lib=g
```

If you use the Suprtool2 interface to invoke Suprtool, you need the following MPE commands to force Suprtool2 to run the call-converted copy of Suprtool:

```
:setjcw suprtool2filecommand = 2
:setjcw suprtool2lib         = 2
:file suprtool = suprtool.cc.disc
```

For Omnidex versions starting with 3.00 you can use call conversion by running Suprtool in the following manner:

```
:run suprtool.pub.robelle;xl="xl.pub.disc"
```

If you encounter any difficulties running Suprtool with call conversion, please call us for assistance.

OmniQuest

Dynamic Information Systems Corporation (DISC) also has a product called OmniQuest, which allows you to qualify records and prompt the user for selection criteria.

In order to allow OmniQuest to work properly, Suprtool needs to use only IMAGE Intrinsics. To do this, you just need to put the following Set command prior to the Base command in your Suprtool and Omniquest command script.

```
>set privmode off
```

Due to a conflict between the Robelle sort and the OmniQuest product, Suprtool's fast sort algorithms must be turned off. This may be done with the following Set command:

```
>set sortfast off
```

If you encounter any difficulties running Suprtool with OmniQuest, please call us for assistance. If you do not turn off Set sortfast off, you will likely receive the error Data Memory Protection Trap.

Suprtool with TRANSACT

TRANSACT is one of the Rapid products sold by HP. A TRANSACT program that serially reads and selects records from a dataset can easily be modified to use Suprtool. To convert a TRANSACT program to use Suprtool, follow these steps:

1. Add a FILE=*SUPRFILE(READ) statement to the SYSTEM statement.
2. Remove any MATCH register statements (use the Suprtool If command instead).
3. Change the FIND(SERIAL) *dataset* to FIND(SERIAL) *SUPRFILE.
4. Use the Suprtool Extract command to select the fields from the LIST= statement.
5. Delete any SORT= statements (use the Suprtool Sort command instead).
6. Use Suprtool to build the SUPRFILE.

The following example shows the changes to a TRANSACT program. A TRANSACT program that serially reads a dataset might look like:

```
system cust, base=cust.database("READ",5);
 << set up to select only cust-out = "YES" >>

move (cust-out)="YES";
set(match) list(cust-out);

find(serial) customer-detail,
    list=(cust-name,cust-addr,cust-nbr,cust-out),
    sort=(cust-nbr),
    perform=p100-process-customer;

end cust;
```

The TRANSACT program is modified to look like:

```
system cust,
       file=*suprfile(read),
       base=cust.database("READ",5);

find(serial) *suprfile,
    list=(cust-name,cust-addr,cust-nbr,cust-out),
    perform=p100-process-customer;

end cust;
```

If this program is a batch program, you would insert the following MPE and Suprtool commands before you invoke the TRANSACT program:

```
!run suprtool.pub.robelle
>base cust.database,5,read
>get customer-detail
>if cust-out="YES"
>extract cust-name,cust-addr,cust-nbr,cust-out
>sort cust-nbr
>output suprfile
>exit
!run tranprog.pub
```

If this program is used on-line, you would invoke Suprtool using the Suprtool2 interface (see the Calling Suprtool user manual). To avoid file name conflicts, on-line programs should use a job temporary Suprtool output file. For TRANSACT to read a temporary file, the asterisk in the file name is required.

If you use the LIST= statement, you have to use the Suprtool Extract command to specify the desired fields. The order of the fields in the Extract command must match the order of the fields in the LIST= statement. The Extract may be omitted if the TRANSACT program reads all of the fields in the dataset, but the program must be recompiled if the dataset is changed.

The Suprtool If command replaces the MATCH register of the TRANSACT program. You can still use the MATCH register to do further selection (e.g., from other datasets).

A FIND(Chain) can be modified to use Suprtool, but you must verify that your program actually runs faster (in many cases it will not). To change a FIND(Chain) to use Suprtool, use the same steps as FIND(SERIAL).

XSORT and RPG

XSORT is a program which simulates the System 34 - #3 sort. XSORT jobs can be converted to use Suprtool with few or no problems. The following steps must be done to convert from XSORT to Suprtool:

1. Change any *Omit Record* statements to use the Define command and a non-equal expression in the If command.
2. Change any *Include Record* statements to use the Define command and the If command.
3. If an *Address Sort* is used, Set Firstrec 1 and use Output xxx,num.

The Set Firstrec 1 is required with KSAM files. This forces Suprtool to start record numbers at 1 regardless of the setting in the KSAM Key file.

QUERY Program

QUERY can't produce a report from an MPE file. Because the output from Suprtool is an MPE file, it is not usually feasible to use Suprtool to improve the speed of

QUERY reports. However, if a small percentage of records are selected, you can use the following method to improve the speed of QUERY reports:

1. Create an empty copy of the database (either build the database using DBSCHEMA or use DBUTIL to empty an existing copy).
2. Suprtool extracts the records and Puts them to the copy of the database.
3. QUERY reports on all the entries in the copy database.

SRN Chronos Dates

Software packages from Software Research Northwest use a date format known as Chronos, which stores the date and time in a compressed 6-byte format. In Suprtool, you can specify this format by using the Srnchronos date attribute:

```
>item box-timestamp,date,srnchronos
```

Once you have used the Item command to identify Chronos dates, you can do the following:

1. Select on these dates using the $date and $today functions. For example, to select records with a timestamp before today:
   ```
   >if   box-timestamp < $today
   ```
2. Extract these dates using $date and $today.
3. See the date attribute in the Form command.
4. Pass the date attribute to STExport by using the Link option of the Output command.
5. Format these dates into a meaningful value by using STExport.

Year 2000 Solutions with Suprtool

For this discussion we'll ignore the fact that the twenty-first century really only started in 2001. ☺

Suprtool often has to process dates in both the twentieth and twenty-first centuries. If you include the century in your dates, Suprtool should behave as most users expect. If you do not include the century in your dates, how Suprtool behaves will be dependent on your specific application and data.

What If I Have Four-Digit Years?

If your dates have four-digit years, Suprtool should work as expected. Selection based on the $today or $date features will select dates in both the twentieth and twenty-first centuries. Dates that do not collate correctly (e.g., mmddccyy) will not be accepted by Suprtool's If command for relative selection (e.g., <, <=, >, or >=). If you have these date formats you can use the $stddate function, converts any date format to a ccyymmdd type date.

Suprtool, as it has always done, will continue to sort dates based on their numeric value and not on any implied date order.

In Suprtool 4.0, we introduced some new command parsing features that let you control how Suprtool parses the year of the $date function. You can either use two-digit years by applying a cutoff rule or you can force all years to be specified as four digits.

What does Set Date Cutoff do?

Date Cutoff tells Suprtool what century to use when Suprtool generates a constant date value from the $date function.

Before version 4.0, Suprtool would assume 19 for the century for any user-specified $date with a two-digit year. For example:

```
>item date-field,date,ccyymmdd
>if date-field <= $date(40/12/26)
```

Previously the $date function would convert the user specified $date to 1940/12/26 in order for it to be compared to the date-field format of CCYYMMDD. Now with Set Date Cutoff xx, Suprtool assumes 20 for the century if the two-digit year specified in the $date function is less than the value of Set Date Cutoff. For example:

```
>set date cutoff 50
>item date-field,date,ccyymmdd
>if date-field <= $date(40/12/26)
```

Suprtool in this case assumes the full $date to be 2040/12/26, because the 40 in $date is less than the 50 in Set Date Cutoff.

The default value of Set Date Cutoff is 10.

Stddate and Set Date Cutoff

When $Stddate has to convert from a date with only a two-digit year, the conversion to the four-digit year will use the value of Set Date Cutoff when converting the date.

For example,

```
>get sales-detail
>set date cutoff 15
>def new-ship-date,1,4,double
>item ship-date,date,mmddyy
>ext order-no / sales-amount
>ext new-ship-date = $stddate(ship-date)
>out salesinfo,link
>xeq
```

In this example, if any ship-date has a year of 14 or less, then the century applied to the new-ship-date field will be 20. Ship-dates with a year of 15 or more will have a century of 19 applied.

What does Set Date ForceCentury do?

Set Date ForceCentury On will not allow a yy date to be entered in the $date function, it will force the user to enter a full ccyy date.

```
>set date forcecentury on
>item date-field,date,ccyymmdd
>if date-field >= $date(98/12/10)

Error: You must specify the century or Set Date ForceCentury off
```

The default value for Set Date ForceCentury is off.

What If I Have Two-Digit Years?

If you have dates with two-digit years, there are two main solutions to making your application ready for the Year 2000:

1. Convert all of your date data to use four-digit years and modify your programs to process four-digit years, or
2. Assume that certain dates are in the twentieth century and some in the twenty-first (this is usually called date windowing).

The first solution requires that you change all Suprtool Item commands for two-digit years to a four-digit year format. If you have not already done so, you may want to isolate all of these Item commands in a single file per input source (e.g., one file for every dataset in every database in your application or just one file for every database). You can nest use-files, making this approach even easier (e.g., having one database use-file that then includes each dataset use-file with a list of Item commands). You may also want to use Suprtool to assist you in changing your actual data from two-digit years to four. See "Can Suprtool Convert Two-Digit Years to Four Digits?" on page 101 for more details.

If you do not include the century in your dates (the second solution above), you will have the following problems:

1. Selecting dates in yymmdd format will not produce the expected results in relative operations (e.g., <, <=, >, or >=). You will need to change all of your If commands to use the $stddate function.
2. Sorting dates that include both 20th and 21st century dates, will not collate the way most users expect, whether with Suprtool, the COBOL sort verbs, or HPs sort tools. This is because Suprtool, and all HP-supplied tools, sort based on the numeric value of a date. To make this work correctly within Suprtool, you will need to use the $stddate function in an Extract command to generate a date with a four-digit year, then sort on this new date field with another Suprtool task.

What Is Wrong with Two-Digit Years?

Currently the date format of yymmdd collates (sorts) correctly if the date is not beyond December 31, 1999. Given the current date of 981210, numerically this is less than next year whose date value is 991210.

At the turn of the century dates in the yymmdd format (or yymm) will no longer sort correctly because the value of December 10, 2000 (001210) is less than 981210.

Consequently, if we have a date beyond 1999, stored in yymmdd format, a relative operation such as:

```
>if date-field >= $date(98/12/10)
```

will not find the date of December 10, 2000. You will need to use the $stddate function to make this task work correctly.

```
>if $stddate(date-field) >= $date(98/12/10)
```

How Do $Today and $Date Work?

Suprtool's date functions ($date and $today) are a short-hand method of generating a numeric constant. So, a date selection like:

```
>item invoice-date,date,YYMMDD
>if invoice-date < $today
```

is exactly the same as:

```
>if invoice-date < 980401                    {on 1st April, 1998}
```

Suprtool does record selection on the *numeric* value of the field and not on the implied date value. If we move the calendar ahead to January 1, 2000 and do the same commands as above, the result would the same as if you had typed:

```
>if invoice-date < 000101                    {on 1st January, 2000}
```

If you have some invoice dates from the previous century (e.g., 990101 for December 1st, 1999), they will not be selected.

Will Suprtool Generate an Error for Two-Digit Year Dates?

Sometimes.

Because dates beyond 1999 will not collate properly for the YYMMDD and YYMM formats, starting in version 4.0.11 the If command produces an error if the year specified in a $date or $today function is greater than 1999 and the date format is YYMMDD or YYMM, and you are performing a relative operation (e.g., <, <=, >, or >=).

```
>item enddate, date, yymmdd
>if   enddate >= $date(*+4/*/*)          {21st. century date}
                           ^
Error:  Cannot use a date beyond 1999 for this format
```

Suprtool returns this error by default, but you can override it with the following set command:

```
>set date ifyy2000error off
```

This tells Suprtool to allow the previously described relative operations and suppress the error message. While you can override the error checking, the behavior of $today and $date is not changed.

How Do I Use $Today and $Date with yymmdd Dates?

If you need to have Suprtool select dates in YYMMDD format with $Today or $Date, you need to use one of the following solutions:

1. Change the date storage format to include the century in all datasets and data files, so you can use the following item command:
   `>item invoice-date,date,CCYYMMDD`
2. Use the $stddate function that adds the century component to dates in a ccyymmdd format in a J2 container.

Also see "Case 1: Converting a J2 Date from yymmdd to ccyymmdd" on page 101 and "Case 2: X6 yymmdd Data to X8 ccyymmdd" on page 103 for more specific details on converting two-digit-year date formats into four-digit-year date formats.

aammdd Date Format

The aammdd date format was developed by James Overman of HP for use in the MM3000 product. This format is only available for the X6 data-type.

The aammdd format is similar to yymmdd, but the year portion of the date use a combination of numbers and letters of the alphabet to represent years beyond 1999.

By substituting a letter of the alphabet in the first position of the year, we can extend a six-digit date and also ensure that the dates collate correctly. For example,

YY of AAMMDD	CCYY
A0 - A9	2000 - 2009
B0 - B9	2010 - 2019
C0 - C9	2020 - 2029

Because letters are greater than numbers in the collating sequence we can ensure that aammdd dates beyond 1999 will order correctly.

Suprtool also supports other date formats with this two-digit year representation. These formats are aamm, mmddaa and ddmmaa.

Invalid Dates

The If command has a $invalid function to find all invalid dates for a particular field. An invalid date is any number of a particular date format whose date equivalent cannot be found on the calendar. For example, a date with a month of 99 will be considered invalid.

```
>base store.demo
Database password [;]?
>get d-sales
>item deliv-date,date,ccyymmdd
>if $invalid(deliv-date)
>out baddates,link
>xeq
```

Can Suprtool Convert Two-Digit Years to Four Digits?

Suprtool is capable of converting dates from one format to another using a variety of Suprtool features. We will show how Suprtool can convert common dates without the century to those that have the century included. While Suprtool can convert your data, it is up to you to change your programs. Adager, a third-party program for changing Image database structures, has the ability to change date fields. Suprtool can convert data in Image databases, flat files, self-describing files and KSAM files.

Case 1: Converting a J2 Date from yymmdd to ccyymmdd

The $stddate function can convert six-digit date formats to ccyymmdd. But what if all the dates are not actually dates, but some dates are filled with 9s as a flag to an application?

Consider this dataset with two date fields, J2 items and in the date format yymmdd.

```
Database: STORE.DB.GREEN
   D-SALES          Detail    Set 5
      Entry:                    Offset
         CUST-ACCOUNT          Z8       1  (!M-CUSTOMER)
         DELIV-DATE            J2       9
         PRODUCT-NO            Z8      13  (M-PRODUCT)
         PRODUCT-PRICE         J2      21
         PURCH-DATE            J2      25
         SALES-QTY             J1      29
         SALES-TAX             J2      31
         SALES-TOTAL           J2      35
   Capacity: 602 (14)  Entries: 10  Highwater: 10  Bytes: 38
```

First, we need to know and understand our data. Are there any invalid dates? If so, does the value have some other logical meaning?

```
>get d-sales
>item deliv-date,date,yymmdd
>item purch-date,date,yymmdd
>if $invalid(deliv-date) or $invalid(purch-date)
>list
>xeq
>GET D-SALES (9) >OUT $NULL (0)
CUST-ACCOUNT    = 10010          DELIV-DATE      = 999999
PRODUCT-NO      = 50513001       PRODUCT-PRICE   = 19220
PURCH-DATE      = 999999         SALES-QTY       = 2
SALES-TAX       = 2691           SALES-TOTAL     = 21910

>GET D-SALES (10) >OUT $NULL (1)
CUST-ACCOUNT    = 10010          DELIV-DATE      = 125213
PRODUCT-NO      = 50513001       PRODUCT-PRICE   = 19220
PURCH-DATE      = 1              SALES-QTY       = 2691
SALES-TAX       = 21910          SALES-TOTAL     = 21910

IN=10, OUT=2.  CPU-Sec=1.  Wall-Sec=1.
```

In this example, we see two records that do not contain proper dates. The first record contains all 9s, which is probably used as some sort of flag. We may need to add 99 in front of these dates.

But the second record is obviously wrong. We can use Dbedit to correct this before converting the other dates. We need to know our data to properly convert to a new date format.

Once all the incorrect dates are fixed, we can start converting. We can add a prefix of 19 or 20 to all the appropriate dates by using the following Extract statement. Please note that we are updating this directly. In case we need to redo this task, we only convert those dates that have not yet been converted. In this example we set the cutoff year to 30 so any dates before 30 will have 20 as the century and the others will have 19.

```
>get d-sales
>set date cutoff 30
>item purch-date,date,yymmdd
>item deliv-date,date,yymmdd
>if not $invalid(purch-date) and not $invalid(deliv-date)
>update
>ext purch-date = $stddate(purch-date)
>ext deliv-date = $stddate(deliv-date)
>xeq
```

We have now converted all the J2 yymmdd dates to a ccyymmdd format and added the correct century to the date.

Case 2: X6 yymmdd Data to X8 ccyymmdd

The following Suprtool task shows how you can generate a new file to put into a new database with dates in a different format.

Consider the deliv-date and purch-date fields of the D-Sales dataset:

```
Database: STORE.DBOLD.ACCOUNT
   D-SALES          Detail    Set 5
      Entry:                    Offset
         CUST-ACCOUNT         Z8      1  (!M-CUSTOMER)
         DELIV-DATE           X6      9
         PRODUCT-NO           Z8     15  (M-PRODUCT)
         PRODUCT-PRICE        J2     23
         PURCH-DATE           X6     27
         SALES-QTY            J1     33
         SALES-TAX            J2     35
         SALES-TOTAL          J2     39
   Capacity: 611 (13)  Entries: 15  Highwater: 15  Bytes: 42
```

You want to convert to a date format with room for a cc at the beginning of the deliv-date and purch-date:

```
Database: STORE.DB.ACCOUNT
   D-SALES          Detail    Set 5
      Entry:                    Offset
         CUST-ACCOUNT         Z8      1  (!M-CUSTOMER)
         DELIV-DATE           X8      9
         PRODUCT-NO           Z8     17  (M-PRODUCT)
         PRODUCT-PRICE        J2     25
         PURCH-DATE           X8     29
         SALES-QTY            J1     37
         SALES-TAX            J2     39
         SALES-TOTAL          J2     43
   Capacity: 608 (16)  Entries: 0  Highwater: 0  Bytes: 46
```

In order to convert these dates, you need to be able to put either a 19 or 20 in front of the yymmdd date, depending on the value of the year. Before you can do either of these you must confirm, once again, that you have no invalid dates.

```
>base store.dbold
>get d-sales
>item deliv-date,date,yymmdd
>item purch-date,date,yymmdd
>if $invalid(deliv-date) or $invalid(purch-date)
>list
>xeq
```

Once you have confirmed that there are no invalid dates you can start converting the dates that you have. Because there are two date fields in this dataset you must be careful to add the appropriate century for the proper field. For this example, assume that if a year is less than 1950 then the century should be 20.

You can easily convert each date by processing each field separately by using an intermediate self-describing file:

```
>base store.dbold,1
Database password [;]?

>get d-sales
>set squeeze off
>item deliv-date,date,yymmdd
>if deliv-date >= $date(1950/01/01)
>out sales01,link
>ext cust-account
>ext "19"
>ext deliv-date / sales-total
>xeq
IN=15, OUT=14.  CPU-Sec=1.  Wall-Sec=5.
```

Now insert 20 to the century for the appropriate records:

```
>base store.dbold
>get d-sales
>if deliv-date < $date(1950/01/01)
>ext cust-account
>ext "20"
>ext deliv-date / sales-total
>out sales01,link,append
>xeq
```

Now you can convert the other field from the flat file, (sales01) and add a century to the purch-date field:

```
>reset
>base                                          {close the open database}
>in sales01
>item purch-date,date,yymmdd
>if purch-date >= $date(1950/01/01)
>set squeeze off
>out sales02,link
>ext cust-account / product-price
>ext "19"
>ext purch-date / sales-total
>xeq
IN=15, OUT=15.  CPU-Sec=1.  Wall-Sec=1.
```

Because you extracted all 15 records you know you do not have any records with the purch-date field that need to be updated with a "20".

Now you can insert the records into the new database:

```
>base store.db
>in sales02
>put d-sales
>xeq
```

Now you have converted two dates from an X6 format to an X8 format.

Case 3: Different Date Formats X6 MMDDYY Data to X6 YYMMDD

The following Suprtool task shows you how to convert a date in a self-describing file from mmddyy to yymmdd format.

Consider the following self-describing file with the deliv-date and purch-date fields:

```
File: SALES04.DATA.ACCOUNT   (SD Version B.00.00)
   Entry:                     Offset
      CUST-ACCOUNT         Z8      1
      DELIV-DATE           X6      9       <<MMDDYY>>
      PRODUCT-NO           Z8     15
      PRODUCT-PRICE        I2     23
      PURCH-DATE           X6     27       <<MMDDYY>>
      SALES-QTY            I1     33
      SALES-TAX            I2     35
      SALES-TOTAL          I2     39
   Limit: 115  EOF: 15  Entry Length: 42  Blocking: 97
```

You want to convert these two dates to a data format of yymmdd before adding a century in front of the year. This can be easily accomplished by defining each sub part of the date and extracting those parts in the new order. You can use string addition to re-order the field in one pass.

```
>in sales04
>def new-deliv,1,6
>item new-deliv,date,yymmdd
>def deliv-date-mm,deliv-date[1],2
>def deliv-date-dd,deliv-date[3],2
>def deliv-date-yy,deliv-date[5],2
>def new-purch,1,6
>item new-purch,date,yymmdd
>def purch-date-mm,purch-date[1],2
>def purch-date-dd,purch-date[3],2
>def purch-date-yy,purch-date[5],2
>ext cust-account
>ext new-deliv = deliv-date-yy + deliv-date-mm + deliv-date-dd
>ext product-no / product-price
>ext new-purch = ext purch-date-yy + purch-date-mm + purch-date-dd
>ext sales-qty / sales-total
>out sales05,link
>xeq
```

You now end up with a file that looks like this:

```
File: SALES06.DATA.ACCOUNT   (SD Version B.00.00)
   Entry:                     Offset
      CUST-ACCOUNT         Z8      1
      NEW-DELIV            X6      9       <<YYMMDD>>
      PRODUCT-NO           Z8     15
      PRODUCT-PRICE        I2     23
      NEW-PURCH            X6     27       <<YYMMDD>>
      SALES-QTY            I1     33
      SALES-TAX            I2     35
      SALES-TOTAL          I2     39
   Limit: 115  EOF: 15  Entry Length: 42  Blocking: 97
```

You then add the century to these fields as described above.

Year 2000 Testing

Testing software to see if it works in the year 2000 and beyond is difficult. Currently there are three software tools that change the software date for specific sessions or applications, making it easier to test software for year 2000 compliance. The four tools are:

1. Hourglass from Allegro Consultants, Inc.
2. Time Machine from SolutionSoft Systems, Inc.
3. Setdate from the HP Jazz machine on the Web
4. TimeWarp/3000 from Omnisolutions

All versions of Suprtool work with Hourglass, but versions previous to 4.0.13 did not work with Time Machine, Setdate, or TimeWarp. This version of Suprtool now works with all these tools. Note that this change only applies to non-expiring versions of Suprtool. Expiring versions of Suprtool (such as trials, pre-releases, and products obtained through VARs) do not work with Time Machine or Setdate.

Performance Issues

HP e3000 sites use Suprtool because it provides access to their data many times faster than they are used to, and because Suprtool enables them to perform time-consuming DP functions easily, with a few simple commands. The typical Suprtool task consists of extracting some data for a report, then feeding the Suprtool output file into the final report program. For example, you might fill a Quiz subfile with the subset of data needed, then report that subfile with Quiz.

Native Mode *and* Compatibility Mode

Suprtool is available in native mode for MPE/iX and compatibility mode for MPE V. You can compare Suprtool's *serial read speed* for TurboIMAGE datasets and MPE files against other programs, both NM and CM. On MPE V systems, Suprtool has always shown a 2 to 10 times performance gain in such serial reads. On MPE/iX systems, Suprtool can make serial reads up to 8 times faster, but the improvements are typically higher on MPE V than on MPE/iX. That is because Suprtool can bypass most of the MPE V operating system, but cannot bypass the MPE/iX virtual page manager.

Here is one MPE/iX test that shows what is possible. The test scans a large detail dataset that is absent from memory, just looking at each record, not writing an output file. This measures raw serial read capability. Suprtool is 2.5 times faster than a standard NM program and uses 1/7th the CPU time. When the dataset is fully in memory, Suprtool is able to complete the task 9.3 times faster than an NM program calling DBGET mode-2.

Comparison of elapsed time, dataset not in memory

CPU time	NM program	115 seconds
	Suprtool	17 seconds
Wall time	NM program	120 seconds
	Suprtool	47 seconds

Comparison of elapsed time, dataset in memory

CPU time	NM program	109 seconds
	Suprtool	12 seconds
Wall time	NM program	112 seconds
	Suprtool	12 seconds

CPU-Bound?

Suprtool uses the technology of multirecord access (NOBUF/MR) to achieve large reductions in CPU overhead, 1/8th that of serial DBGET in many cases. On a CPU-

bound MPE/iX system, such significant savings mean that you can run Suprtool when other programs would be an intolerable burden.

Sort Speed

Suprtool/V uses the HP sort intrinsics, so sort times should not show much change. Suprtool/iX uses its own set of high-speed sort intrinsics, unless one of the following is true:

- Set Sortfast is Off (the default is On).
- Native Language Support is enabled (either automatically via the NLDATALANG JCW or explicitly via Set NLS).
- Any of the sort fields are defined as type CHAR.

Generally, Suprtool's internal sort algorithm is faster than HP's sort algorithm.

Analyzing Performance Data

It is better to test Suprtool with your own database and your own application needs, rather than trust to a "generic" performance test. The ideal test is an actual production report whose bad performance is causing you a problem. Then if you obtain improvements by using Suprtool you know you can achieve better speed in practice as well as in tests.

Use Suprtool as a front end to your problem report, producing a small extract file containing just the fields and records needed for your final program. Once you get that working, consider linking in data from other files or datasets using Suprlink. For comparison purposes, run the Suprtool test at the same time as you would normally run the original program. Comparing a standalone midnight run against a mid-day run does not give valid results, nor does comparing two runs in succession (the second run benefits on MPE/iX, since the files are already in memory at that time).

You have several choices when deciding how to extract the data for your report. One user had a report that consumed 18 hours on his Series 955. Since the site had only a short window for batch processing each night, the job would spill over into the prime shift. By the time the report did complete, it was a day late and a new report was running. His boss was considering going to a 960 to reduce the job time. The programmer tried Suprtool as a front end, but it only reduced the time to 15 hours. When he called Robelle in disappointment, we investigated his application. He had used the Chain command because he had a TurboIMAGE search item for his key values and thought following such a search path would be fastest.

We suggested a serial Get command instead of a random-access Chain command and the job time fell to 4.5 hours--much easier to schedule. Get was faster than Chain, even though it had to look at every entry in the dataset. Why? Get reads many contiguous records with each access to MPE, while Chain must call TurboIMAGE to retrieve each chain entry. Since the entries on the chain are seldom in the same disc block, Chain generated many more disc accesses, and they were randomly distributed over the disc space. In this case, he was selecting 11% of the dataset (261,230 out of 2,307,685 entries). Chain had to do about 424,000 random disc reads, while Get did only 14,471 serial reads, a reduction of 97%. However, the Chain command would have been faster if he had selected less than 15,000 entries. You can use the Set Statistics On command to see how many *disc reads* a Get command uses; if that is less than the number of *entries* selected, Get will usually be faster than Chain.

Suprtool Performance Hints

Not all portions of the Suprtool product give equal performance boosts. The primary technology of Suprtool is large NOBUF/MR reads and writes. By asking the operating system for many records at once, instead of invoking the system software for each record, Suprtool reduces the CPU overhead of the task dramatically. This technology is present when you use the Input and Get commands, but not the Chain command (for some tasks the Get command must revert to using the slower standard database reads, but it always prints a warning).

The If and Extract commands are powerful for reducing a large input file into a small output file that contains just the data desired. They are reasonably efficient, but usually consume as much processor time as the serial read itself. The If command does partial evaluation of expressions. Therefore, you should place the tests that are most likely to fail at the start of the expression, and place the table lookups (which are slower) at the end. This minimizes overhead. Using Extract reduces the size of the output record, and greatly improves sort times (if all sort keys are extracted).

The Delete, Put, and Update commands are implemented by calling the corresponding TurboIMAGE intrinsics: DBDELETE, DBPUT, and DBUPDATE. A simple way to improve Delete, Put, and Update performance is to use Set Lock 0 (see below), but this is rarely appropriate when on-line users need access to the database. There is another option that improves the speed of Deletes, Puts, and Updates, but it comes with many warnings (see "Defer" on page 254).

The Dbedit Subsystem, an interactive editor for dataset entries and chains of entries, does not use NOBUF/MR, so is strictly a convenience, not a performance aid.

Suprlink is a tool for linking together fields from several files by a common key value. It links sorted files that Suprtool extracts from IMAGE, KSAM and MPE files. It uses NOBUF/MR techniques and gives you a performance improvement over random database accesses in many cases. However, if you have to re-sort the data many times and the records are large, Suprlink may not help you. Performance will be optimized if you extract only the fields you need from each input source, not the entire record.

Speed Demon is an intrinsic library for 3GL programs that provides a high-speed replacement for TurboIMAGE serial "gets". The NM version returns records at the same speed that Suprtool does with the same low CPU overhead. The MPE V version has half the speed of Suprtool at retrieving records. Consider calling Speed Demon in your programs when you need to process more than 50% of the records in a file, since you eliminate the need to read Suprtool's output file.

Occasionally we will put performance tips on our web site that are timelier than what can be achieved in a manual. So please also visit:

http://www.robelle.com/tips/st-performance.html

for the latest performance issue and tips.

Obtaining Accurate Measurements

The performance results obtained from Suprtool, like any program, vary depending on the record sizes, number of records, size of sorts, and the current efficiency of your application. Measuring performance on any computer system is a challenge, but especially so on MPE/iX with its demand-paged virtual memory and complex scheduler. Some tests do not show such a marked improvement as the examples described here. That is why we recommend that you test Suprtool against your own database.

Suprtool works best when selecting a small subset of a large dataset. The larger the dataset and the smaller the percentage extracted, the better the performance improvement. Once you have selected a task to measure, you have the difficulty of making a fair comparison. This can be trickier than it seems. For example, on MPE/iX if you run your regular task first, then the Suprtool task, you will almost certainly find the Suprtool task faster, just because the first task has made most of the data resident in memory.

Here are the guidelines that we follow in order to obtain performance measurements that we can be confident of:

1. We do every test at least five times. It is rare for each test to be identical.
2. All MPE/iX performance tests are sensitive to the amount of the file that is in memory. We use the Klondike Nugget (now available from Lund Performance Solutions) to measure how much of our test files are in memory before we start each test.
3. We start each test by flushing all files out of memory. We do this with the unsupported Fflush tool. This program is difficult to obtain (we cannot give you a copy) and you probably need a new version for each new release of MPE/iX. You might try getting a copy from your HP SE.
4. Tests are run in the middle of the night and we ensure that there are no other sessions or jobs running during each test.

Performance Summary

Suprtool *is* a performance tool. Not because every Suprtool command is always fast, but because performance is one of our first concerns when adding new features to Suprtool. We always look for the fastest way to do any task, even if that means it can't be quite as flexible as other software tools.

Suprtool Functions

Over the years we've grown the if/extract function list, to the point where it needs its own documentation for both the if and extract commands. Functions can be used in both the if and extract commands, however, some functions were sometimes written to either be specifically used in either if or extract, but typically have an application in both.

String/Byte Functions

Suprtool has a number of functions that work on String/Byte type fields. In all cases the target and source of these functions are byte type fields and are treated as strings internally to Suprtool.

$TRIM (Works on byte type fields)

Purpose is to remove spaces from the beginning and the end of a field.

If Usage:

```
>if $trim(last_name) = "ARMSTRONG"
```

This means that Suprtool will qualify "ARMSTRONG", " ARMSTRONG", " ARMSTRONG" etc.

Extract Usage (target: Byte type fields)

```
EXT byte_target=$trim(byte_source)
```

Example:

```
DEF lastname,1,16,byte
EXT lastname=$trim(last_name)
```

Data Examples Before and After:

```
>IN STRINGS.NEIL.GREEN (0) >OUT $NULL (0)
LASTNAME        =   ARMSTRONG
ext lastname=$trim(lastname)
>IN STRINGS.NEIL.GREEN (0) >OUT $NULL (0)
LASTNAME        = ARMSTRONG
```

$LTRIM (Works on Byte type fields)

Purpose is to remove spaces from the left side of the field.

If Usage:

```
if $ltrim(first_name) = "NEIL "
```

Extract Usage (target: Byte type fields)

```
ext byte_target=$ltrim(byte_source)
```

Example:

```
DEF lastname,1,16,byte
EXT lastname=$ltrim(last_name)
```

Data Examples before and after:

```
>IN STRINGS.NEIL.GREEN (0) >OUT $NULL (0)
LASTNAME        =   ARMSTRONG
ext lastname=$ltrim(lastname)
>IN STRINGS.NEIL.GREEN (0) >OUT $NULL (0)
LASTNAME        = ARMSTRONG
```

$RTRIM (Works on Byte type fields)

Purpose is to remove spaces from the right side of the field.

If Usage:

```
if $rtrim(first_name) = "Neil"
```

Extract Usage (target: Byte type fields)

```
extract byte_target=$rtrim(byte_source)
```

Example:

```
def lastname,1,16,byte
ext lastname=$rtrim(last_name)
```

Data Examples:

```
>IN STRINGS.NEIL.GREEN (0) >OUT $NULL (0)
LASTNAME        =   ARMSTRONGbb
>IN STRINGS.NEIL.GREEN (0) >OUT $NULL (0)
LASTNAME        =   ARMSTRONG
```

It is difficult to show what $RTRIM does in a simple instance, however, it is perfect for constructing/merging two separate fields into on as in:

```
ext fullname=$rtrim(first-name) + " " + $trim(last-name)
```

Data Result

Neil Armstrong

$UPPER (Works on Byte-type fields)

Purpose is to make all relevant bytes "Upper" case.

If Usage:

```
if $upper(first_name) = "NEIL"
```

Extract Usage (target: Byte type fields)

```
extract byte-target=$upper(byte-source)
```

Example:

```
def lastname,1,16,byte
ext lastname=$upper(last_name)
```

Data Examples:

```
>ext lastname=$upper(lastname)
>IN STRINGS.NEIL.GREEN (0) >OUT $NULL (0)
LASTNAME        = ARMSTRONG
```

$LOWER (Works on Byte-type fields)

Purpose it to make all relevant bytes "Lower" case.

If Usage:

```
if $lower(first_name)  = "neil"
```

Extract Usage:

```
ext byte_target=$lower(byte_source)
```

Example:

```
 def newfirstname,1,16,byte
 ext newfirstname=$lower(first_name)
```

Data Examples:

```
>ext lastname=$lower(lastname)
>IN STRINGS.NEIL.GREEN (0) >OUT $NULL (0)
LASTNAME        = armstrong
```

$PROPER (Works on Byte-type fields)

Purpose is to make the relevant bytes either Upper or Lower Case in an intelligent manner. It will Upshift the first character in a field and anything following a space or dash.

If Usage:

```
if $proper(first_name) = "Neil"
```

Extract Usage:

```
ext byte_target=$proper(byte_source)
```

Example:

```
def fullname,1,30,byte
ext fullname=$proper(first_name)
```

Data Examples:

```
>ext fullname=$proper(fullname)
>IN STRINGS.NEIL.GREEN (0) >OUT $NULL (0)
FULLNAME        = Neil Armstrong
```

$SPLIT (Works on Byte-type fields)

Purpose is to extract a string into a byte type field that begins at a certain character instance and ends and a second character instance. Commonly used to reformat name fields and read .CSV files.

If Usage:

Not that commonly used.

Extract Usage:

```
extract field=$split(field_from,start_pos,instance,end_pos,instance)
define acct-x,1,12,byte
define fullname,1,30
ext acct-x=$split(record,first,",")
ext fullname=$split(record,",",",",")
```

Data Examples:

```
Source Data of record              acct-x result
123456789,"Neil Armstrong",435     123456789
```

Data Examples:

```
>IN STRINGS.NEIL.GREEN (0) >OUT $NULL (0)
ACCT-X          = 123456789
>IN STRINGS.NEIL.GREEN (0) >OUT $NULL (0)
FULLNAME        = "Neil Armstrong"
```

$FINDCLEAN (Works on Byte-type fields)

Purpose is to find any specified character(s) in any byte string fields anywhere in a given byte field.

If Usage:

```
clean "^10"
if $findclean(e-mail)
```

Records will qualify if a Line Feed is in the e-mail field.

$CLEAN (Works on Byte-type fields)

Purpose was to "clean" and remove unwanted characters from any byte-string fields in any point in the field. Typically used to removed un-printable characters from byte fields.

If Usage: (Not commonly used)

```
clean "N"
if $clean(first_name)=" eil"
```

Extract Usage:

```
extract byte_target=$target(byte_source)
```

Example:

```
clean "N"
set cleanchar " "
extract byte_target=$clean(byte_source)
```

Data Examples:

```
Source: Neil
Target:  eil
```

Example, remove Line Feed from E-mail Address field and shift data to the left:

```
Clean "^10"
set cleanchar "<null>"
ext e-mail=$clean(e-mail)
```

$TRANSLATE (Works on Byte-type fields)

Purpose to translate any character from a byte type field to any defined byte, primarily to obfuscate data and was primarily designed for the extract command.

If Usage:

```
translate "A:C"
if $translate(Full_Name)="Neil Crmstrong" { not commonly used }
```

Extract Usage:

```
translate "A:C"
translate "B:D"
translate "a:f"
extract New-byte-field=$translate(byte-field)
```

Data Examples: (Using above code)

```
Source Data:     Result Data
Barry Armstrong  Dfrry Crmstrong
```

$JUSTIFYL (Works on Byte-type fields)

Purpose to justify the field on the left side of the byte field.

If Usage:

```
if $justifyl(Full_Name)="Neil Armstrong"
```

Extract Usage:

```
extract New-byte-field=$justifyl(byte-field)
```

Data Examples: (Using above code)

```
Source Data:             Result Data
     Barry Armstrong     Barry Armstrong
```

$JUSTIFYR (Works on Byte-type fields)

Purpose to justify the field on the right side of the byte field.

If Usage:

```
if $justifyr(Full_Name)="Neil Armstrong"
```

Extract Usage:

```
extract New-byte-field=$justifyr(byte-field)
```

Data Examples: (Using above code)

```
Source Data:        Result Data
Barry Armstrong            Barry Armstrong
```

$LEADZEROB (Works on Byte-type fields)

Purpose is to add leading zeros to a particular byte field with an option to justify the data.

If Usage:

```
if $leadzerob(numdata,J)="0000006"
```

Extract Usage:

```
extract NewNumField=$leadzerob(byte-field,J)
extract NewNumField=$leadzerob(byte-field,N)
```

Data Examples: (Using above code)

```
      6              0000006
     4               000004
```

$RESPACE (Works on Byte-type fields)

Purpose is to Respace a particular byte field, it takes multiple spaces and reduces them down to one, great for fixing up addresses.

If Usage:

```
if $respace(NAME)=”Neil Armstrong”
```

Extract Usage:

```
extract NewDescField=$respace(byte-field)
extract NewDescField=$respace(byte-field)
```

Data Examples: (Using above code)

```
  This  is   some data   This is some data
  This  is   some data     This is some data
```

$LENGTH (Works on Byte-type and numeric fields)

Purpose of the $length function is to return the length of the specified byte field as if the field has had the $rtrim function applied or the position of the rightmost non-space character.

If Usage:

```
if $length(NAME) >= 15
```

Extract Usage:

```
Define LenField,1,4,double
extract LenField=$length(byte-field)
```

$ETOA

Purpose is to convert Ebcidic to Ascii, for a byte type field.

Extract Usage:

```
Extract $etoa(char-field)
```

$ATOE

Purpose is to convert Ascii to Ebcidic, for a byte type field.

Extract Usage:

```
Extract $atoe(char-field)
```

$etoa and $atoe cannot be used in if command, nor do they put a result in a field, they just extract the converted byte-type field.

String Addition

Suprtool is capable of doing string addition, again this is with byte type fields, you can do add various byte type fields together and even the results of various functions from above.

Extract Usage:

```
extract target_byte_field = byte-field1 + " " + byte-field2
extract FullName = $trim(first_name) + ' ' + $trim(last_name)
```

Example:

```
extract FullName = $trim(first_name) + ' ' + $trim(last_name)
```

Data Result:

```
>IN STRINGS.NEIL.GREEN (0) >OUT $NULL (0)
FULLNAME        = "Neil Armstrong"
```

Numeric Functions

There are a number of functions that are used to assist and help perform some arithmetic operations.

$TRUNCATE

Purpose is to not round a given result or arithmetic expression or number. Suprtools default behaviour is to round a result, the $truncate function will change that behaviour.

If Usage:

```
if $truncate((qty * price) / 100 = 100
```

Extract Usage:

```
extract new_price=$truncate((qty * price) / 100)
```

$ABS

Purpose is to return the absolute value of a given number.

IF Usage:

```
if $abs(credit-field)=5000
```

Extract Usage:

```
extract newnum=$abs(credit-amt)
```

$TOTAL

Purpose is to provide a running total for any numeric field and deposit the result into a packed-decimal field.

IF Usage:

Not commonly used in if

Extract Usage:

```
define mytotal,1,18,packed
ext mytotal=$total(sales-amount)
```

$SUBTOTAL

Purpose is to provide a running total for any numeric field and deposit the result into a packed-decimal field. The target packed-decimal field is reset to 0 on a control break on the specified sort field.

IF Usage:

Not commonly used.

Extract Usage:

```
define target,1,18,packed
sort sort-field
ext target=$subtotal(fieldtototal,sort-field)
```

Example of $TOTAL and $SUBTOTAL

```
>in file1sd.suprtest
>def mytotal,1,14,packed
>def mysubtotal,1,14,packed
>sort char-field
>ext char-field
>ext int-field
>ext mytotal=$total(int-field)
>ext mysubtotal=$subtotal(int-field,char-field)
>list
>xeq
>IN FILE1SD.SUPRTEST.GREEN >OUT $NULL (0)
CHAR-FIELD      = 11111          INT-FIELD       = 1111
MYTOTAL         = 1111
MYSUBTOTAL      = 1111

>IN FILE1SD.SUPRTEST.GREEN >OUT $NULL (1)
CHAR-FIELD      = 22222          INT-FIELD       = 2222
MYTOTAL         = 3333
MYSUBTOTAL      = 2222

>IN FILE1SD.SUPRTEST.GREEN >OUT $NULL (2)
CHAR-FIELD      = 22222          INT-FIELD       = 2222
MYTOTAL         = 5555
MYSUBTOTAL      = 4444

>IN FILE1SD.SUPRTEST.GREEN >OUT $NULL (3)
CHAR-FIELD      = 33333          INT-FIELD       = 3333
MYTOTAL         = 8888
MYSUBTOTAL      = 3333

>IN FILE1SD.SUPRTEST.GREEN >OUT $NULL (4)
CHAR-FIELD      = 33333          INT-FIELD       = 3333
MYTOTAL         = 12221
MYSUBTOTAL      = 6666

>IN FILE1SD.SUPRTEST.GREEN >OUT $NULL (5)
CHAR-FIELD      = 33333          INT-FIELD       = 3333
MYTOTAL         = 15554
MYSUBTOTAL      = 9999
```

$COUNTER

Purpose is to provide a running counter starting from 1, and the target/result field is a double integer.

IF Usage:

Not commonly used.

Extract Usage:

```
define mycounter,1,4,double
ext mycounter=$counter
```

$SUBCOUNT

Purpose is to provide a running counter starting from 1, and the target/result field is a double integer. The counter is reset when the control-break occurs on the specified sort field.

IF Usage:

Not commonly used

Extract Usage:

```
define mysubcount,1,4,double
sort customer-no
ext mysubcount=$subcount(customer-no)
```

Examples for $counter and $subcount:

```
in file1sd.suprtest
def mycount,1,4,double
define mysubcount,1,4,double
sort char-field
ext char-field
ext mycount=$counter
ext mysubcount=$subcount(char-field)
list
xeq
>IN FILE1SD.SUPRTEST.GREEN >OUT $NULL (0)
CHAR-FIELD      = 11111          MYCOUNT         = 1
MYSUBCOUNT      = 1

>IN FILE1SD.SUPRTEST.GREEN >OUT $NULL (1)
CHAR-FIELD      = 22222          MYCOUNT         = 2
MYSUBCOUNT      = 1

>IN FILE1SD.SUPRTEST.GREEN >OUT $NULL (2)
CHAR-FIELD      = 22222          MYCOUNT         = 3
MYSUBCOUNT      = 2

>IN FILE1SD.SUPRTEST.GREEN >OUT $NULL (3)
CHAR-FIELD      = 33333          MYCOUNT         = 4
MYSUBCOUNT      = 1

>IN FILE1SD.SUPRTEST.GREEN >OUT $NULL (4)
CHAR-FIELD      = 33333          MYCOUNT         = 5
MYSUBCOUNT      = 2

>IN FILE1SD.SUPRTEST.GREEN >OUT $NULL (5)
CHAR-FIELD      = 33333          MYCOUNT         = 6
MYSUBCOUNT      = 3
```

$SIGNED

When the target of an extract conversion is a packed or display- type field, Suprtool always converts positive values to a neutral number. To ensure that expressions with positive values have a positive result, use the $signed function:

IF Usage:

Not commonly used:

Extract Usage:

```
extract packed-field=$signed(int-field)
extract display_field=$signed(dbl-field / 10)
```

$LEADZEROZ (Works on Display-type fields)

Purpose is to add leading zeros to a particular byte field with an option to justify the data.

If Usage:

```
if $leadzeroz(numdata,J)="0000006"
```

Extract Usage:

```
extract NewNumField=$leadzeroz(disp-field,J)
extract NewNumField=$leadzeroz(disp-field,N)
```

Data Examples: (Using above code)

```
      6              0000006
     4               000004
```

$LENGTH (Works on Byte-type and numeric fields)

Purpose of the $length function is to return the length of the specified byte field as if the field has had the $rtrim function applied or the position of the rightmost non-space character.

If Usage:

```
if $length(num-field) > 15
```

Extract Usage:

```
Define LenField,1,4,double
extract LenField=$length(num-field)
```

Arithmetic Operations

+ - * / mod

Suprtool has the ability to perform any and all Arithmetic functions on numeric fields.

If Usage:

```
if sales-total <> (sales_qty * unit_price) + sales_tax
```

Extract Usage:

```
ext sales-total=(sales_qty * unit_price) + sales_tax
```

Conversion/Formatting

$NUMBER

Purpose is to convert a freeform number, to a number that can be put into an actual numeric field. The target must be numeric, and the source field must be a display type field, even though it can have nonnumeric characters in it.

If Usage:

```
if $number(disp-field)=500
```

Extract Usage:

```
define disp-field,byte-field,display  {define byte as display with same
length}
define newdouble,1,4,double
ext newdouble=$number(disp-field)
```

Data Examples:

$5.00 ---> 500

Notes: Suprtool will try and match the decimal places of the defined by the item command of the numeric target to the decimal places in the actual data and handle the data properly. For example, if the target field has an item command definition of two decimal places, Suprtool will handle the data Raw data accordingly:

```
    Raw Data     Result of $number
    5.1             5.10
    5.123           5.12
    5.139           5.14
```

$EDIT

Purpose is to format a number or byte-string field to a particular format using an editmask similar to the syntax found in Cobol into a target byte-type field. Primarily designed to help format data for list and/or reports.

If Usage:

```
if $edit(somefield,"$$$$.99-")="$5.00" {Not commonly used}
```

Extract Usage:

```
ext byte-field=$edit(dbl-field,"$$$$.99-")
ext byte-field=$trim($edit(dbl-field,"$$$$.99-"))
```

Data Examples:

500 $5.00

Other Functions

$LOOKUP

Purpose is to determine if a key value has been loaded into memory with options to reference associated data.

If Usage:

if $lookup is capable of determining if a key value exists and also compare against a field specified with the data field.

```
table tablename,key,file,filename
if $lookup(tablename,key)
table tablename,key,file,filename,data(data-field)
if $lookup(tablename,key,data-field) = inputsrcdatafield
```

Extract Usage:

```
table tablename,key,file,filename,data(data-field)
extract target-field=$lookup(tablename,key,data-field)
```

$READ

Purpose is to expand the command line limit for an if command beyond 256 characters.

If Usage:

```
if $read
-number=1 and
-number=3 and
-//
```

$INRECNUM

Purpose is to reference the input record number for a given input source.

If Usage:

```
if char-field=”55555” and $inrecnum=123
```

Extract Usage:

```
Def recnum,1,4,double
Ext recnum=$inrecnum
```

Date Functions

$TODAY

Purpose is to determine the current date.

If Usage:

```
If date-field=$today(-1)
if date-field=$today
```

Extract Usage:

```
ext target-date=$today
```

The date-field and target-date must be defined as having a particular date format (item command) and in a proper container for that particular date type.

$DATE

Purpose is to determine a given date.

If Usage:

```
if date-field=$date(*/*/*)
```

Extract Usage:

```
ext date-field=$date(*/*/*)
```

Date-field must be defined as having a particular date format (item command) and in a proper container

$INVALID

Purpose is to determine if a given date is valid or not.

If Usage:

```
item check-date,date,ccyymmdd
if $invalid(check-date)
```

Extract Usage:

Not typically used in extract.

$STDDATE

Purpose is to convert from any supported Suprtool date in any container to CCYYMMDD in a double integer container. For date formats with only two digit years Suprtool will look at the value of Set Date Cutoff in order to determine what century to convert it to, whether it be 19 or 20. You can also specify that the $BOM (beginning of the month) or the $EOM, be returned to transform any given date to the beginning of the month (first) or the end of the month.

If Usage:

```
item my-yymmdd-date,date,yymmdd
if $stddate(my-yymmdd-date) = 20141213
```

Extract Usage:

```
item my-yymmdd-date,date,yymmdd
define new-ccyymmdd-date,1,4,double
ext new-ccyymmdd-date=$stddate(my-yymmdd-date)
ext new-ddyymmdd-date=$stddate(my-yymmdd-date,$EOM)
```

$DAYS

Purpose is to convert any supported Suprtool date to a number of days since 4713BC or Julian Day format. Main purpose in if is to compare two dates and get the

diffference. Main purpose in extract is to typically transform a date to be so many days away from the current date value.

If Usage:

```
if ($days(entry-date) - $days(order-fulfill-date)) >=30
```

Extract Usage:

```
extract new-date=$stddate($days(entry-date) + 10)
```

$MONTH

The if / extract commands can utilize a function called $Month, which will add a given number of months to a given date in the format of ccyymmdd or yyyymmdd. You can also specify the date to be returned as the $BOM, (beginning of the month), or $EOM, which will return the End of the Month for the calculated month for the given date.

For Example:

```
    In somefile
    Item mydate,date,ccyymmdd
    Def targetdate,1,4,double
    Ext targetdate=$month(mydate,+4)
```

The above task will take the field mydate and add four months to it. Suprtool will check if the date is valid and adjust the date within reason. For example, if the given month for mydate has 31 days and the day is 31, and the month mydate becomes when the date is added to has only 30 days. The date will be adjusted to have the 30th for the day.

If Usage:

```
If $month(entry-date,+10) = 20161213
```

Extract Usage:

```
extract new-date=$month(entry-date,+10)
extract new-date-bom=$month(entry-date,+10,$BOM)
extract new-date-eom=$month(entry-date,+10,$EOM)
```

Suprtool Commands

General Notes

Do not enter the > sign when typing commands.

When you run Suprtool, it prompts for commands with a ">" character and reads command lines from the standard input device. Suprtool commands contain a command name which may include one or more optional parameters that are each separated by commas.

In this chapter, we describe the Suprtool commands in alphabetic order. Each command name is followed by its minimal abbreviation in brackets. For example: [D] for Define and [DU] for Duplicate.

Most Suprtool commands work within the context of the input file. In general, the Base, Chain, Get, and Input commands must be entered before other commands. Once the input source has been specified, commands can be entered in any order.

Abbreviating

You may shorten the command name to any substring that uniquely defines the command. For example, Form can be abbreviated as FO or F, since there are no other commands that start with "F". Duplicate, however, can be abbreviated only to DU, since there is also a Define command in Suprtool.

```
>i sdfile                                {Input command}
>l                                       {List command}
>x                                       {Xeq command}
```

Uppercase or Lowercase

You can enter the letters in either uppercase or lowercase because Suprtool upshifts everything in the command line except literal strings within quotes (e.g., "abc"). These two commands are identical:

```
>EXTRACT QTY
>extract qty
```

Multiple Commands per Line

You can enter several commands on a single line, if you separate them with semicolons. An entire "task" can be placed on one input line.

```
>in old;out new;xeq
>get m-customer;if cust-status<>10,20,30;list;x
```

Caution: Suprtool cannot distinguish between several commands on one line and several commands entered on several lines. This is not a problem when using Suprtool in batch, as Suprtool stops executing when an error occurs. But when using Suprtool interactively, specifying multiple commands separated by a semicolon, Suprtool keeps on parsing the rest of the line after it finds an error. For example, if you misspell the "fldname" when you type the following,

```
>get dsetname; if fldname="value";del;out filename;xeq
```

Suprtool sends you an error message with the typo, but continues with the rest of the command line. This has the effect of deleting all the entries in the dataset. It is risky to type Xeq on the same command line as any other command unless:

1. You are not concerned with the consequences if you make a mistake (e.g., any "extract" task should be safe).
2. You don't make any mistakes.

The usual reason for putting all the commands on one line, including the Xeq, is to permit repetition of the task by using the Before command. It is not necessary to type everything on one line because with the Listredo command, Suprtool allows you to pick and choose any of your last 1000 commands.

Continuation

The maximum command that may be specified in the Info= parameter of the Run command is 80 characters.

The maximum *physical* command line is 256 characters. You may enter commands on multiple input lines by putting an "&" continuation character at the end of the line. The maximum total command length is 256 characters. The separating comma in commands is not optional. Should your If command exceed 256 characters, use the Table command, or $read.

```
>in sdfile
>if status="20" and &                    {continue the If command}
    state="AZ","CA","OR"                {select several states}
>output outfile
```

Comments on Command Lines

You may also use MPE's :COMMENT command to enter comment lines.

Comments may appear at the end of any command line, as long as they are surrounded by curly braces. Many of the examples in this manual show comments at the end of command lines. You can enter a comment as the only item in a Suprtool command line. When you enter continued command lines, the comment can appear before or after the continuation character:

```
>{The following task extracts all customer records for}
>{   the different customers we are interested in.}
>in customer               {input self-describing file}
>if status = "10" or &     {prepaid status}
    status = "20" or &     {current status}
    status = "30"          {arrears status}
>output outfile           {output to a disc file}
>exit                     {execute the task and exit}
```

STREAMX

STREAMX is a product from VESOFT that permits you to build flexible job streams. STREAMX contains a complete programming language with loops, prompts, and parameter substitution. A problem arises when trying to enter

comments into a Suprtool batch job that will be submitted with STREAMX. Suprtool uses the {...} pair to delimit comments. STREAMX uses these same characters for expressions. Similarly, Suprtool uses question marks as part of pattern-matching in the If command, and STREAMX uses question marks for questions.

You cannot change Suprtool's comment or pattern-matching characters, but you can change the expression and question characters in STREAMX. The following example changes the STREAMX expression characters from {...} to ~...~, and the question character from ? to `:

```
!job example,user.acct
::setquestion `
::setbraces ~~
::prompt date datevar = "Enter date to select: "
!run suprtool.pub.robelle
>base store                          {input database}
>get d-sales                         {and dataset}
>if deliv-date=~datevar~ &           {STREAMX substitutes actual date}
>and product-code == "??X?3@"
>output mpefile
>exit
!tell ~hpjobname~,~hpuser~.~hpaccount~;Example done.
!eoj
```

MPE Commands

If Suprtool doesn't recognize the command you have entered, it tries to interpret it as an MPE command. Suprtool also interprets any command line beginning with a colon (:) as an MPE command. Only the commands that MPE allows in "break" are allowed in Suprtool/V. This feature can be used to establish :File commands for the input or output files, to :PURGE the output file to avoid a duplicate file name error, and to include :COMMENT lines. For example:

```
>:comment sort custfile by custnum
>input custfile
>key 1,10                                {no colon on the next command}
>purge sortcust                          {make sure that no duplicate file error occurs}
>output sortcust
>exit
```

MPE/iX Commands

Suprtool/iX executes any MPE command (e.g., Run), UDCs, and command files. **Caution:** programs that suspend, instead of terminating, are not killed by the HPCICOMMAND intrinsic.

Calculator

Any command line beginning with an equal sign (=) is treated as a calculator expression. This feature can be used to compute blocking factors and do other calculations without the need of an electronic calculator. For help, type =?.

Control-Y Interrupt

You can interrupt a Suprtool task with the Control-Y key (hold down Control while striking Y). Suprtool responds by telling you how much work it has done (IN=,OUT=,etc.) and asks if you wish to stop. Hit the Return key to continue or type YES to stop the task. Even though you abort the task, your output file is saved (although it may be empty if you stop before the sort phase is over).

If you use Control-Y during a Put, Delete, or Update operation, the database locks are still in effect. Do not walk away from your terminal without answering YES or NO to the "stop" question. If you are Deleting records, you should continue the task unless you don't want the output file. With the output file, you can recover the records back into the dataset by using the Put command.

When you hit Control-Y, MPE must fix Suprtool's stack markers so that you resume where the interrupt occurred. With MPE V, we have received reports of invalid-stack-marker aborts in Suprtool, when using Control-Y while reading a dataset. This is probably because the stack-marker format changed in MPE V, and Control-Y has trouble finding the proper one when Suprtool is in Privileged Mode. There is not much you can do except to avoid Control-Y while reading a dataset.

Error Recovery

There are several errors that can occur during a Suprtool "task":

1. The output file fills up.
2. There are invalid packed-decimal or zoned-decimal digits in the input record.
3. There is a duplicate key when writing to a KSAM file.
4. An attempt is made to delete a master dataset entry which still has associated detail entries.
5. Suprtool tries to Put a record to a detail dataset that does not have an associated master entry.
6. An attempt is made to add a master dataset entry which already exists.
7. An attempt is made to delete a dataset record that has changed.

For errors 4 through 7:

If Set Ignore is Off (default), Suprtool prints an error message and stops processing the input file. For KSAM errors, the standard file information display is printed. For IMAGE errors, Suprtool prints whatever information is available on the record that caused the DBDELETE, DBPUT, or DBUPDATE error (when Set Dumponerror On).

The information from IMAGE errors may include the input record number, the output record number and the data values of the record. Suprtool uses the current options of the List command to print the data values. If no List command was specified, Suprtool uses the List defaults.

To only see a summary of IMAGE errors that occurred, turn Set Dumponerror Off. To continue processing, even when errors occur, turn Set Ignore On.

Add Command [AD]

The Add command currently is available only for Oracle access on HP-UX.

Base Command [BA]

Opens a specified database. Once you have it open, you can extract data from datasets, delete entries, load new entries into datasets, use the schema to define fields for MPE files (Input file = set), and do on-line Editing of the database (Edit allows add, change, delete, list, modify of individual fields and entries).

To access a remote DS database, specify the *system* name in the Base command. To access a remote NS database, use a :File command.

To close the current database, use Base without parameters.

BASE [*system*#]*basename* [,*mode* [,*password*]]

(Default: *system*=logon, *mode*=1, *password*=Creator)

System is the DS name of a remote HP e3000 (optional). *Basename* is the name of your database. *Mode* is the DBOPEN mode that you want (i.e., mode-1 for shared updates, mode-5 for shared read-only), and *password* is the DBOPEN database password. When the *password* is included in the Base command, it is always upshifted. Use the ? option to specify lowercase passwords.

Suprtool opens the Base, which remains open until you do another Base command, a Reset Base, or a Reset All, even if you do several extracts from the database.

Examples

The first example shows a typical Suprtool task. A dataset in the Store database is read and a subset of the entries are sorted into a disc file:

```
>base store,5,READER                {open for read access only}
>get d-sales                        {select an input dataset}
>if sales-total>10000               {choose a subset of all entries}
>sort cust-account                  {sort by account number}
>output salesout                    {output has same structure}
>exit                               {   as the d-sales dataset}
```

Our next example opens the database with the Creator password (which is the default). No output file is produced; instead, we produce a formatted listing of the input dataset:

```
>base store                         {use the Creator password}
>form sets                          {you cannot remember the names}
>get d-inventory                    {   of the datasets in Store}
>list                               {print the dataset fields formatted.}
>exit                               {  no output file is created}
```

In session mode, this command would prompt for the database password. If none is entered, or Suprtool is not running interactively, the Creator password is used by default (;). Because the open mode was not specified, the database is opened in mode-1.

Database Passwords in Batch

In batch mode, it is necessary to specify a database password without the password being echoed on the job stream listing. A special database password is provided to allow for this. When a question mark "?" is used as the database password, Suprtool prompts for the password on the next physical input line without echoing. This occurs in batch mode or in session mode. For example:

```
>base store,5,?                          {Suprtool prompts for the password.}
Password >                              {Password is on this line, but it won't}
>get d-inventory                        {show on job stream listing or terminal.}
>exit
```

File Commands

You can use a :File command to re-direct a database name to another group and account in Suprtool. For example:

```
:run suprtool
>:file dbx=db23.data.test
>base dbx
>get cust-master
```

Because the remote database feature generates a :File command, you cannot combine these two features. If you specify a remote system in the Base command, the specified *basename* must be precise. Any existing :File command will have been overruled. When Suprtool closes a remote database, any :File command for the database is reset.

DS/3000 Database Access

Suppose that you wished to obtain the total of the ON-HAND-QTY and the UNIT-COST of the inventory in a remote DS-database. The following Suprtool example would obtain the totals:

```
>base newyork#store.data,5,READER       {NEWYORK database}
>get d-inventory
>total on-hand-qty                      {produce a total of this field}
>total unit-cost                        {total this field too}
>exit                                   {no output file is created}
```

You must have already done the necessary :DSLINE and :REMOTE HELLO commands. This is the only way to read from a remote database in Suprtool. You cannot use the "Remote Database Access" facility of IMAGE. Suprtool does a :File command for the base name before calling DBOPEN.

```
:file basename=basename; dev=system#disc.
```

NS/3000 Database Access

There are two restrictions to accessing a database on a remote NS-system:

1. You cannot use the *system* name of the Base command.
2. You cannot specify the NS/3000 *node specification* as part of the database name.

The only way to access a database on a remote NS-system is to use a :File command:

```
:dsline mrp.go.hp
:remote hello user.account,group
:file store=store:mrp.go.hp
:run suprtool.pub.robelle
>base store
>get m-customer
>output mcust
>xeq
```

Jumbo Datasets

Due to limitations in MPE/iX, Suprtool cannot open remote Jumbo datasets. Suprtool will fail with an error similar to that showed as below:

```
Error: Unable to open MPE file /ACCT/GROUP/DATASET.001
```

Privileged File Violation

When trying to access a database on a remote computer with Suprtool, the Get command may fail with the error "Privileged File Violation (FSERR 45)". This is the result of changes made to NS to improve security for remote file access. Apparently, some customers complained to HP that a program on the local system should not be permitted to FOPEN a privileged file (such as an IMAGE dataset) on a remote system unless the user on the remote system has PM capability.

There are three workarounds to this problem:

1. Set Privmode Off in the Suprtool task, before the Base command. This forces Suprtool to use slow DBGETs instead of fast NOBUF/MR access.
2. Log on to the remote system as a user with PM capability and proper read/write access to the database (such as Manager.Sys). Or run a program on the remote system such as GOD from VESOFT, granting the remote user the required capability and file access privileges.
3. Log on to the remote system as you have done in the past, but run Suprtool on the remote system instead of the local system. That is, run Suprtool on the same computer as the database resides, and direct the output file back to the local computer. An added advantage of this method is that only selected database records have to be passed across the NS connection.

For many tasks this improves performance dramatically. Suppose you want to select transactions for one month from a yearly history dataset. If Suprtool is run on the local system and the database resides on the remote system, then all the records in the dataset must be passed across the NS connection, just to have most of them rejected by Suprtool's selection criteria. On the other hand, if Suprtool is run on the same system as the database, then only the selected records have to be passed across the NS connection.

This is the solution that we recommend. If you do not have Suprtool on all your production systems, you can use one of the other workarounds, or call us for a right-to-copy license.

Open Modes

Base does not allow mode-7 for exclusive read access, but it does allow mode-4 for exclusive update while others read. Mode-4 disallows changes while you are extracting from the database. Mode-3 for fully exclusive access is tolerated, but causes Suprtool to use slow DBGETs to extract from datasets. Mode-3 should only be used when you need to do a Delete or Put command with Set Defer On.

Suprtool allows you to open a second database in the Put command when you are copying from one database to another.

Before Command [B]

Repeat any combination of the previous 1000 command lines, with or without editing.

BEFORE [*start* [/ *stop*]]
[*string*]
[ALL | @]

(Default: redo previous line)

(BQ=redo without change)

The Before command allows you to modify the commands before it executes them. If you don't need to change them, use BQ or Do.

The Before command uses Qedit-style Control characters for modifying the commands. The default mode is to replace characters. To delete use Control-D, and to insert use Control-B. If you prefer HP-style modify (D, R, I, and U), use the Redo command instead of Before.

Examples

```
>listf @.soruce                          {"source" is not spelled right}
NON-EXISTENT GROUP.  (CIERR 908)
>Before                                  {redo most recent command}
listf @.soruce                           {last command is printed}
         our                             {you enter changes to it}
listf @.source                           {the edited command is shown}
                                         {you press Return}

>listredo -10/
>before 5                                {redo 5th command in stack}
>bef 8/10                                {redo 8th through 10th}
>b listf                                 {redo last Listf command}
>b listf temp                            {redo "listf temp" command}
>b @temp                                 {redo last containing "temp"}
>before -2                               {redo command before previous one}
>before -5/-2                            {redo by relative lines}
```

Modify Operators

If you wish to change any characters within the line, the modify operators are the regular Control Codes used in Qedit:

Characters	Action
Any printing characters	replace the ones above
Control-D plus spaces	deletes columns above.
Control-B	puts you into "insert before" mode.
Control-A	starts appending characters at the end of line.
Control-A, Control-D, plus spaces	deletes from the end.
Control-T	ends Insert Mode, allowing movement to a new column.
Control-G	recovers the original line.
Control-O	specifies "overwrite" mode (needed for spaces).

Persistent Redo

Redo commands can be saved in a permanent file and can therefore be used from another session. You can use the **Set redo** command to specify a filename to save your redo commands. Please see "Redo" on page 267 for details.

Chain Command [C]

Selects an IMAGE dataset and a search path as the input source for the next extract. Unlike the Get command, the Chain command reads records by using the standard IMAGE intrinsics. For detail datasets, the Chain command does a DBFIND on the specified search path and uses DBGET mode-5 to read the records. For master datasets, the Chain command uses DBGET mode-7. You must have read access to all fields in the dataset entry. Only one Chain, Get, or Input command is allowed per extract task.

CHAIN *setname,search-field=value [,...]*

CHAIN *setname,search-field=table-name*

When using Chain, the size of the output file defaults to the size of the Chain dataset. If the dataset is large and the selected subset is small, you should use the Numrecs command to reduce the size of the output file.

Selection by Individual Values

Suppose you want to find all the sales records for customer account "1234". Assuming that cust-account is a search field in the d-sales dataset, you would use this:

```
>base store,5,READER                    {open for read access only}
>chain d-sales,cust-account="1234"      {read the sales records for customer 1234}
>out out1
>xeq
```

If you want to select sales records for more than one customer, you would use:

```
>chain d-sales,cust-account="12345","67890","39201"
>out out2
>xeq
```

Values with Decimal Places

You can specify values with decimal places for search fields with implied decimal places. For example,

```
>item unit-cost,decimal,2               {two implied decimal points}
>chain d-inventory,unit-cost=10,10.5,10.75
>out out3                               {select records for 1000, 1050, and 1075}
>xeq
```

Selection with a Table

You can specify the records to read using a table. The Table command may appear before or after the Chain command. Use this option when you have a file of key values to search for:

```
>{select the sales records that match the values in custfile}
>chain d-sales,cust-account=sales-table
>table sales-table,cust-account,file,custfile
>out out4
>xeq
```

Combining Chain with If

The Chain command selects an input dataset and a set of key values to search for. To specify additional selection criteria, use the Chain command to specify the key values and the If command for the additional selection criteria:

```
>{select records for customer 1234 where the sales-qty is }
>{ greater than 100}
>chain d-sales,cust-account="1234"
>if sales-qty > 100
>out out5
>xeq
```

B-trees and Chain

The Chain command only takes advantage of B-trees if the BtreeMode1 setting is turned on at the root level. This setting means that all DBFIND mode-1 calls will utilize the B-tree.

You can turn BtreeMode1 on by doing the following:

```
:run dbutil.pub.sys
>>set BtreeMode1=on
```

The Chain command allows the partial key retrieval on X- and U- type items on the master dataset, and implicitly on any associated detail dataset. For example,

```
>base cd.db.data,5,reader
>chain a-songs,song-title="w@"
>list
>xeq
Warning:  Using DBGET for the input records
>GET A-SONGS (289) >OUT $NULL (0)
SONG-TITLE      = we only come out at night

>GET A-SONGS (56) >OUT $NULL (1)
SONG-TITLE      = where boys fear to tread

IN=2, OUT=2.  CPU-Sec=1.  Wall-Sec=1.
```

As previously mentioned, we can search a detail dataset by using the implicit B-tree or the search item that has a B-tree attached to the associated master.

```
>chain d-songs,song-title="w@"
>list
>xeq
Warning:  Using DBGET for the input records
>GET D-SONGS (37) >OUT $NULL (0)
CD-NO      = 2                  CD-TITLE    = Mellon Collie
ARTIST     = Smashing Pumpkins  SONG-NO     = 10
SONG-TITLE = we only come out at night
>GET D-SONGS (28) >OUT $NULL (1)
CD-NO      = 2                  CD-TITLE    = Mellon Collie
ARTIST     = Smashing Pumpkins  SONG-NO     = 1
SONG-TITLE = where boys fear to tread

IN=2, OUT=2.  CPU-Sec=1.  Wall-Sec=1.
```

Third-Party Indexing

If MPE/iX third-party indexing is enabled, the Chain command takes advantage of third-party indexes in the specified dataset. The *search-field* parameter can be any IMAGE index or any third-party index. Only X- and U-type fields are supported. Use the Form command to see which index fields Chain will accept. The *value* you specify is still restricted to the maximum length of the *search-field*. The following example uses a third-party index to only match customer entries where the last name starts with the letter "A":

```
>chain m-customer,name-last="A@"
>list standard
>xeq
```

Notes

The Chain command is intended to replace a Get and If combination where the primary selection is by key value. In many cases, it is still faster to use the Get command to read the entire dataset than it is to use the Chain command to use search-paths or third-party indexes. Use Set Stat On to compare the performance of the Chain and Get commands.

The dataset search field cannot be a R-type item.

The Chain command always reads the chain values in sorted order by ascending search-value. You do not need to specify a Sort command to have the output file sorted by the search field.

Clean Command [CL]

Specifies what characters to clean when using the $Clean function.

CLEAN [SPECIAL | <string> <range>]

(Default: None)

Suprtool will "clean" or replace all of the characters specified in the Clean command from a byte type field when invoked by the $Clean function. To define what characters that need to be replaced you use the clean command with the character you want to clean in quotes. Since most of the characters that you will need to clean are unprintable, you can enter the decimal equivalent of the character. This is denoted by entering the "^" character in quotes preceding the decimal number of the character you wish to clean.

An example of how easy it would be to clean your database of certain "bad" characters in byte-type fields would be as follows:

```
>base mydb,1,;
>get customer
>clean "^9","^10","^0","^7"
>update
>ext address(1) = $clean(address(1))
>ext address(2) = $clean(address(2))
>xeq
```

The SPECIAL keyword automatically defines Clean characters of Decimal 0 thru to Decimal 31.

```
>base mydb,1,;
>get customer
>clean special
>update
>ext address(1) = $clean(address(1))
>ext address(2) = $clean(address(2))
>xeq
```

You can also specify a range of characters with the following syntax:

```
>base mydb,1,;
>get customer
>clean special,"^128:^255"
>update
>ext address(1) = $clean(address(1))
>ext address(2) = $clean(address(2))
>xeq
```

The above task would clean all byte type fields of any characters from Decimal 0 (Null) to Decimal 31 plus Decimal 128 to Decimal 255.

Removing Bad Characters

You can have the Clean function clean the field, and instead of replacing with a space, the $clean function will essentially shift characters to the left by Setting the CleanChar in the following manner:

```
>Set Cleanchar "<null>"
```

Suprtool will pad the field that was cleaned with the appropriate amount of characters with a space at the end of the field.

Define Command [D]

Defines fields that can be used in the Duplicate, Extract, If, Item, Sort, Table, and Total commands. With Define, you can do selection on ordinary data files using the same kind of readable "expression" that you use with databases. You can also access data fields that are not actually structured as defined in the database (e.g., implicit subfields within an IMAGE field).

DEFINE *field*, *definition*

field is an identifier up to 32 characters long, must begin with a letter, and can consist of letters A through Z, digits 0 through 9, or the following symbols:

```
+ - * / ? # % & @ _ '
```

In the case where the field name is written to a self-describing file, only the first 16 characters are used.

definition can be in two different forms: absolute or relative.

Absolute Definitions

DEFINE *field*, *byteposition*,*sublen* [,*type*] [,*subcount*]

(Default: *type*=BYTE, *subcount*=1)

The *byteposition* is a positive integer giving the byte index where the field starts. The first byte is always number 1, not 0. The *sublen* is the number of bytes in the field. When the *subcount* is 1 (default), the *sublen* is the total number of bytes in the field. When you specify a *subcount*, the *sublen* is the byte-length of each subfield.

See **Data-Types** below for the definition of *type*.

```
>input discfile                          {input from a disc file}
>def qty,14,4,double                     {double integer (PIC S9(9) COMP)}
>def name,5,6                            {character string of 6 bytes}
>sort name                               {sort using the field "name"}
>total qty                               {total all the values of the field "qty"}
>exit
```

Relative Definitions

DEFINE *field, fieldname* [*(subscript)*] [*[offset]*] [,*sublen*] [,*type*]

[,*subcount*]

(Default: *sublen*/*type*=same as *fieldname*)

The *fieldname* is an IMAGE field from the dataset specified in Chain, Get, or Input=*dataset*, or a field from a self-describing file, or another Defined field. Relative definitions are similar to COBOL's Redefine verb.

The *sublen* and *type* are optional. They default to the total byte length and type of the *fieldname*. The (*subscript*) parameter is an optional sub-item index for arrays such as IMAGE compound items like 5J2 or 4X20. The first sub-item is number 1, and if no subscript is provided, Suprtool uses the first sub-item. The *[offset]* parameter is optional and specifies a byte offset from the position that would otherwise be used. This allows you to define fields relative to other fields. The *[offset]* starts at 1 and not at 0 (i.e., FIELD[1] is the first byte of the field).

To define a field that corresponds to the second street-address of m-customer, you would use:

```
>get m-customer
>define city,street-address(2)
>if city="Vancouver"
>list
>xeq
```

Data-Types

Here are the valid *types*:

Type	Description
BYTE	printable ASCII characters
INT/INTEGER	two's complement
DOUBLE	two's complement
REAL	Classic floating-point
LONG	Classic floating-point
IEEE	IEEE floating-point
PACKED	packed-decimal
PACKED*	packed-decimal, last nibble unused
DISPLAY	zoned-decimal numeric field
LOGICAL	unsigned integer
CHARACTER	for Native Language Support

For compatibility with Sort.Pub.Sys, the Define command also accepts Fpoint as the data-type for IEEE numbers.

The following table shows the Suprtool definitions for the IMAGE data-types:

IMAGE Type	Number of Bytes	COBOL Declaration	SUPRTOOL Definition
I1	2	S9(4) COMP	>define a,1,2,integer
I2	4	S9(9) COMP	>define a,1,4,double
I4	8	S9(18) COMP	>define a,1,8,integer
J1	2	S9(4) COMP	>define a,1,2,integer
J2	4	S9(9) COMP	>define a,1,4,double
J4	8	S9(18) COMP	>define a,1,8,integer
U*n*	*n*	A(*n*)	>define a,1,*n*,byte
X*n*	*n*	X(*n*)	>define a,1,*n*,byte
Z*n*	*n*	9(*n*)	>define a,1,*n*,display
P*n*	*n/2*	S9(*n-1*) COMP-3	>define a,1,*n/2*,packed
K1	2		>define a,1,2,logical
K2	4		>define a,1,4,logical
R2	4		>define a,1,4,real
R4	8		>define a,1,8,long
E2	4		>define a,1,4,ieee
E4	8		>define a,1,8,ieee

Data-type Display may have a trailing overpunch sign.

Packed-Decimal Fields

When defining packed-decimal fields, you must convert the number of decimal digits into a byte count. The last digit of a decimal field is *always* used for the sign. There are two data-types for decimal fields: PACKED for those that end on a byte boundary and PACKED* for those that end in the middle of a byte. Here are some example definitions of packed-decimal fields:

Suprtool Definition	Description
>define m,1,2,packed	s9(3) COMP-3, P4 in IMAGE, 2 bytes, 4 nibbles, last is sign digit
>define n,1,2,packed*	s9(2) COMP-3, P4 in IMAGE, 2 bytes, 4 nibbles, last digit is unused
>define p,1,6,packed	s9(11) COMP-3, P12 in IMAGE, 6 bytes, 3 words, 11 digits

Data-Type Warning

The Define command accepts field definitions of any combination of byte-length and data-type. However, many combinations have limited usefulness in Suprtool. In these cases, Suprtool prints a warning. For example:

```
>def field,1,1,integer
Warning:  Length of 1 is of limited use for the data-type INTEGER
```

Examples

The following examples show the various data-types and combinations that are available with the Define command:

```
>define a,11,4,double                {J2 or I2, S9(9) COMP}
>define b,21,2,int                   {J1 or I1, S9(4) COMP}
>define c,3,4,real                   {R2, Classic floating-point}
>define d,5,8,long                   {R4, Classic floating-point}
>define e,21,8                       {character string}
>define f,address(5)                 {fifth occurrence of address}
>define g,address(5)[3],4            {relative offset}
>define h,address(5)[4],2,byte       {middle of the address}
>define i,x[6],4                     {starts at sixth byte of X}
>define j,invoice,6,byte             {redefine field as Byte}
```

Absolute Example

The following example shows the most basic use of the Define command to create a new field definition at an absolute location in the input record.

```
{"amt" is an integer that starts at the 11th byte of file}
>def amt,11,2,int
>if amt > 1000                          {"amt" is now a field we can select on}
>output outfile
```

Absolute Example with Subcount

IMAGE and Suprtool allow fields to be repeated. In the next example, we define an amount field that repeats twelve times (e.g., once for each month of the year). We use a subscript when we want to refer to a specific month.

```
>def amt,11,2,int,12                   {"amt" is an integer that repeats 12 times}
>if amt(5) > 1000                      {we select on the 5th subfield}
>output outfile
```

Relative Examples

Use the simplest form of relative definitions to rename existing fields.

```
>def quantity,qty                      {"quantity" is a more readable name}
>if quantity = "100 "                  {selection on the new field}
```

The Define command copies the byteposition, sublength, and type to the new field, but it does not copy the subcount. Define assumes that you want the first subfield:

```
>def amount,amt                        {amt is 12J2}
>if amount > 1000                      {amount is 1J2 and is the same as amt(1)}
```

Relative Example with Subcount

Because Suprtool defaults the subcount to one, you might want to specify an explicit subcount when giving a new name to an existing field.

```
>def amount,amt,,,12                   {amt is 12J2; same length and type}
>if amount(5) > 1000                   {amount is 12J2; we are selecting for May}
```

Relative Example with Subscript

Use subscripts to define a new field that corresponds to a specific subfield.

```
>def may,amt(5)                        {amt is 12J2}
>if may > 1000                         {may is the fifth subfield}
```

Relative Example with Offset

Many applications define subfields within a larger character field. A common example is a part-number where the first four digits are the warehouse location, the second four digits are the bin number, and the last four digits are a serially assigned number. Use the offset parameter to define new fields that are relative to the start of the part number. The file INVOICES is a self-describing file.

```
>in invoices                           {part is 12 bytes}
>def warehouse,part,4                  {warehouse starts at part}
>def bin,part[5],4                     {bin is second four bytes}
>def release,part[9],4                 {release is the last four bytes}
>if bin = "100"                        {use any field for selection}
```

Note that redefining the digits of a larger numeric field only works when the field is a character field (type X, U, or A).

Relative Example Combining Subscripts and Offsets

If we have ten part numbers combined into one field (e.g., 10X12) we can still define a single field that corresponds to the bin number of one of the parts.

```
>in invoices                              {allparts is 10X12}
>def bin3,allparts(3)[5],4                {we are checking starting at the fifth byte …}
>if bin3 = "100"                          {… of the third part}
```

Notes

The purpose of the Define command is to tell Suprtool to look at a portion of the input record in a certain way. For example, if the record contains ASCII digits in the first ten bytes, Define could be used to assign a field definition to the first six bytes. In this case, the field must be defined as data-type Byte, Char or Display.

The Extract command may be used for data conversion. See the Data Conversion *section of the Extract command for details.*

The Define command **cannot** be used to convert data from one format to another. Using the same example, the Define command could not be used to treat the first six digits of the ten digit ASCII field as Integer, Packed, Real, or any other numeric data format. If the input record contains ASCII digits, Define cannot change them to another data-type.

It may be helpful to think of the Suprtool Define command as similar to a COBOL REDEFINES clause, or a FORTRAN EQUIVALENCE statement.

Delete Command [DEL]

Deletes all entries selected from the input dataset.

DELETE

Delete has no parameters, and can be entered only after the source of input records has been specified using Get or Chain. Delete causes Suprtool to "delete" the input records after they have been read, either all of the input records, or a subset defined by the If command. Delete is not supported for KSAM and MPE files, only for IMAGE datasets.

Examples

```
>:comment Delete and sort old transactions, write to file.
>base store,1,writer                    {open for write access}
>get d-sales                            {dataset to delete from}
>if purch-date<980101                   {select which records to delete}
>delete                                 {ask for deletion}
>sort cust-account                      {sort by account and sort}
>sort purch-date                        {   by date}
>output oldsales                        {output file with deleted and sorted records}
>exit
```

Losing Records

Delete can be combined with other operations, such as select, sort, write to output file, and even a Put to another dataset. Care must be used when combining operations, because such combinations cannot be restarted if the Suprtool "run" is aborted for any reason (including hitting Control-Y and answering "yes"). See "Suprtool and IMAGE" on page 59 for more details on using the Delete and Put commands in combination.

Suprtool deletes records during the input phase. This means that records are deleted before they are sorted. All of the selected input records are deleted before the first output record is written when sorting the input. If you press Control-Y and answer "yes" before a delete task has completed, there may be no way to recover your deleted records.

Improving Performance of Delete with Deferred Output

Suprtool uses the standard Image intrinsic DBDelete to remove entries from a dataset. This makes tasks with a Delete operation slower than the standard MR/NOBUF extracts, but insures structural integrity of the database and that any Third-Party Indexes are kept in sync.

You can reduce the time needed to Delete dataset entries by enabling deferred output for the database. If you open the database in mode-3 (exclusive) and Set Defer On, Suprtool calls DBCONTROL to defer posting of IMAGE writes to disc. The database creator can enable deferred output processing for all users of a database with DBUTIL (enable base for autodefer).

Deferred output can make deletes (and puts) faster, but if the system crashes (i.e., you have a system failure or a nonrecoverable power failure) while you are using deferred output, you must RESTORE your database from tape. You do not have to restore the database if Suprtool stops because of an error or if you abort Suprtool (i.e., break/abort or :abortjob) while deferred output mode is being used.

Omnidex

If you are deleting from a dataset with Omnidex keys, you must have MPE/iX third-party indexing (TPI) enabled in your database, or you must run Suprtool with Omnidex's call conversion facility. To run with call conversion, you must run Suprtool with special parameters.

For Omnidex version 3.0.00 and greater:

```
:run suprtool.pub.robelle;xl="xl.pub.disc"
```

For Omnidex version 2.05.03 to 2.09:

```
:run suprtool.cc.disc;lib=g
```

Warning Message

When deleting records from the input dataset, Suprtool asks for permission to delete all records if there is no If command to select the records to be deleted:

```
Delete all records from the D-SALES dataset [no]:
```

Respond Yes for the task to continue.

In batch mode, Suprtool assumes Yes and keep processing.

Note that if there is no If command, but there is some other command to limit the number of records deleted (e.g., the Numrecs command or a record-range on the Get command), Suprtool still asks the question, implying that all records in the dataset will be deleted. In these cases, only the records that are read, based on the limiting factor, will be deleted.

For interactive approval of each deletion, see the #Delete command of Dbedit.

Notes

For IMAGE manual master datasets, a DBDELETE call fails if the entry to be deleted is a "chain-head" with related detail entries still linked to it. When this happens, Suprtool, by default, prints an error and stops processing the input data. If Set Dumponerror On, the record is printed on $stdlist. If Set Ignore On has been entered, Suprtool continues processing and prints a count of "chain-heads" at the end. Records that cannot be deleted are not included in the output file.

Do Command [DO]

The Do command repeats (without changes) any of the previous 1000 commands.

DO [*start* [/ *stop*]]
[*string*]
[ALL | @]

(Default: repeat the previous command)

Commands are numbered sequentially from 1 as entered and the last 1000 of them are retained. Use the Listredo command to display the previous commands. You can repeat a single command (`do 5`), a range of commands (`do 5/10`) or the most recent command whose name matches a string (`do list`). If you want to modify the commands before executing them, use Redo or Before.

Examples

`>listredo`	{see the previous 20 commands}
`>do`	{do previous command again}
`>do 39`	{do command line 39 again}
`>do 5/8`	{do command lines 5 to 8 again}
`>do list`	{do most recent List command}
`>do show`	{do last starting with "show"}
`>do showjob job`	{do last "showjob job" command}
`>do @job`	{do last containing "job"}
`>do -2`	{do command before previous one}
`>do -7/-5`	{do by relative line number}
`>do 5/`	{do command lines 5 to "last"}

Notes

The Do command can be abbreviated to `,.`, as in MPEX . You cannot use ";" to combine commands on the same line.

Duplicate Command [DU]

By default, Suprtool copies all selected input records to the output file. The Duplicate command determines what to do with duplicate output records. Duplicate records can be discarded, producing an output file without duplicates. Alternatively, you may be interested in seeing the duplicate records, so you can create an output file consisting solely of the duplicate records. When deciding whether an output record is a duplicate, Suprtool either compares the keys only or the entire output record.

```
DUPLICATE     NONE | ONLY
              RECORD | KEYS [num]
              [ COUNT ] [ TOTAL ... ]
```

None

The None option removes duplicate records from the output file. Suprtool compares each output record with the previous output record. If they are not the same, the record is added to the output file. This option corresponds to the former Nodup and Nodupkey options of the Output command.

```
>key 1,4
>duplicate none keys

 Input         Output

1111  10      1111  10
2222  25      2222  25
2222  35      3333  48
3333  48
```

Only

The Only option is the exact opposite of None. Only selects all output records that would not be written by the None option. When the Only option finds a record that duplicates a record already in the set, it writes that duplicate to the output file. Note that the first record is not written to the output file. Here are two examples:

```
>key 1,4
>duplicate only keys

 Input         Output           Input         Output

1111  10      2222  35         1111  10      2222  35
2222  25                       2222  25      2222  42
2222  35                       2222  35
3333  48                       2222  42
```

Record

Suprtool has two methods for comparing output records: Record and Keys. The Record option compares the entire output record. This option can be specified without a sort, but in that case the input file must already be sorted. Note that there are two data fields in the records in the following example, so that a comparison of the entire record yields no duplicates.

```
>duplicate none record
 Input        Output

1111  10     1111  10
2222  25     2222  25
2222  35     2222  35
3333  48     3333  48
```

Keys

The Keys option compares only the sort keys to determine whether an output record is a duplicate. This option requires that at least one sort key be specified.

```
>sort agent                          {sort by agent}
>duplicate none keys
>output agents                       {create roster of agents}
```

The Keys Num option determines the level at which Suprtool compares sort keys. This option controls which duplicate records get included in (or excluded from) the output file.

In the following example we sort by agent and by bill-date (in descending order), but only check for duplicates at the agent level.

```
>sort agent                          {sort by agent}
>sort bill-date,desc                 {sort by date}
>duplicate none keys 1               {only check for duplicate agents}
>output agents                       {create roster of agents}
```

Count

The Count option causes Suprtool to produce a new field in the output record with the number of occurrences of each key value. The count field is called st-count, and is an I2-type field. The Count option can only be used with Duplicate None Keys.

```
>key 1,4
>duplicate none keys count

 Input       Output

1111  10    1111  10  1
2222  25    2222  25  2                {two records for key value 2222}
2222  35    3333  48  1
3333  48
```

Total

The Total option allows up to 15 fields to be subtotaled for each duplicate key. Separate the fields with spaces, not commas. The Total option can only be used with Duplicate None Keys. A new field is created at the end of the output record for each total. Each field is called st-total-*n*:

```
>sort customer-no
>extract customer-no
>duplicate none keys total sales-qty sales-amt
```

The above commands will create a self-describing file with the field customer-no and the total by each customer of the sales-qty in the field st-total-1. Similarly, the field st-total-2 will contain the total sales-amt by customer number.

The following data-types are chosen for each total based on the data-type of the field:

Data-Type	Total Data-Type
Real	R4
IEEE	E4
all others	P28

Please see the P28 Fields section on how to define these fields in Cobol and PowerHouse programs.

Note that for byte fields, there can be only digits in the field. If there are other characters such as "+", "-", or ".", then Suprtool reports an error.

You can use the Link output option to easily see the fields that Suprtool creates. For repeated fields (e.g., 6I2), the first subfield is subtotaled if you don't provide a subscript. You can combine the Count and Total options, but the Count option must appear before the Total option. You cannot combine the Total command with the Total option of the Duplicate command.

```
>key 1,4
>def field,5,2,integer
>duplicate none keys total field

 Input       Output

1111  10    1111  1  10
2222  25    2222  2  60                {25 + 35 = 60}
2222  35    3333  4  48
3333  48
```

Deleting Duplicate Records

There is no direct way to delete duplicate records. Specifying both the Delete and Duplicate commands does not delete all duplicate output records. This is because the Delete command occurs in the input phase, as records are read and selected, but the Duplicate command occurs in the output phase after the sort has taken place. The Duplicate command is not part of the selection process; it is the If command that determines what records are deleted. A common mistake is to specify the Duplicate and Delete commands without an If command. In this case, Suprtool asks your permission to delete *all* input records. Answer "No" to this question to abort the task.

It may be possible to remove duplicates from an input dataset using Suprtool, but only through roundabout methods. These are too case-specific to document here. If you need to do this, call Robelle technical support for assistance.

Self-Describing Files

When you are using the Count or Total options, the sort information is not retained in the output file. This means that if your output file is in LINK format, there are no sort keys. If you are going to use this file from Suprlink, use the BY-clause of the LINK command to manually specify the sort keys.

Non-Self-Describing Files

If the input file is not a self-describing file and you are using Count or Total, you also need to extract all the fields from the file. You can quickly extract the fields by defining a new field that contains the entire record. For example, if your records are 56 bytes long, then you do the following to extract all the fields:

```
>define entire,1,56,byte
>extract entire
```

Output Restrictions

When you are using the Count and Total options, the only Output formats that you can use are Data, Data-Num, Query, and Link. If you want to quickly see the Count or Total results without using a second pass, use the List Standard command.

```
>get dataset
>sort key-field
>duplicate none keys count total data-field
>output result,link
>list standard
>xeq
```

P28 fields in COBOL and PowerHouse

Some versions of COBOL do not allow a field greater than 18 digits. To use the P28 field, you must define the first five bytes of the field as filler. For example,

```
05  ST-TOTAL01.
    10 FILLER                      PIC X(05).
    10 TOTAL-1                     PIC S9(17) COMP-3.
```

PowerHouse users should define this field as follows:

```
Define Total   packed size 14
```

Notes

The option of Only or None must be specified before the option of Record or Keys. Reversing the order causes a syntax error in the Duplicate command.

You cannot combine the Total command with the Total option of the Duplicate command.

It is important to note that if the field being sorted is the result of a $function, then you may not be sorting on the value of the field after the function has transformed the field. For example, the following task may not give you the result you expect:

```
>base store,5
>get d-sales
>def cust-accountx,1,6,byte
>ext cust-accountx = $edit(cust-accout,"zzzz99"
>sort cust-accountx         {sorting on transformed field before it has value from function}
>dup none keys
>output dsales
>exit
```

In this instance you will not be sorting on cust-accountx as transformed by the $edit function, but rather the first six bytes of d-sales. Therefore, it is important to note when you are using extract to transform a field, you will not be sorting on that transformed value.

Therefore, the way to do the transformation would be to either divide into two tasks or if you can logically sort on the field before the transformation in order to achieve the result, so the task could be:

```
>base store,5
>get d-sales
>def cust-accountx,1,6,byte
>ext cust-accountx = $edit(cust-accout,"zzzz99"
>sort cust-account                        {Note sorting on source field}
>dup none keys
>output dsales
>exit
```

or if two tasks are necessary:

```
>base store,5
>get d-sales
>def cust-accountx,1,6,byte
>ext cust-accountx = $edit(cust-accout,"zzzz99"
>output tempsales
>xeq
>in tempsales
>sort cust-acctx
>dup none keys
>output dsales
>exit
```

Edit Command [ED]

Edits an IMAGE/3000 database.

EDIT

The Edit command of Suprtool invokes an independent module of Suprtool called Dbedit. This module permits the user to add, change, list, or delete individual records or "chains" of records from an IMAGE database. This module should be useful for debugging systems, doing data entry on simple systems, and quickly prototyping systems. The Edit command uses the database opened previously by the Base command. Where Suprtool is primarily a serial tool, the Edit module is primarily a tool for chained or keyed access.

Examples

```
>:comment Edit the STORE database.
>base store,1,writer                    {open for write access}
>edit                                   {switch to Dbedit}
#list d-inventory                       {Dbedit prompts with "#"}
#exit                                   {returns control to Suprtool}
>                                       {Suprtool prompts again}
```

Notes

An entire user manual has been provided to describe how to use the Edit feature of Suprtool. See the Dbedit section in the manual for all the details.

Exit Command [E]

Exit Suprtool in one of three ways: quit immediately, suspend entry of commands, or execute the current task then quit.

EXIT [ABORT | SUSPEND | XEQ]

(Default: XEQ)

Users are often frustrated when they exit Suprtool after specifying only part of a task because Suprtool starts processing the task. To exit Suprtool without executing the current task, use the Abort or Suspend options.

Typing Exit with no parameters means Exit Xeq. Suprtool recognizes special command names which specify both the Exit command and an exit option (e.g., ES means Exit SUSPEND).

Exit Abort [EA]

Cancels the current operation and terminates Suprtool. The Exit command without parameters always attempts to perform the task currently specified, while Exit Abort cancels the task and terminates immediately. Thus, Exit Abort is similar to Reset All;Exit with these differences: should Suprtool be executed as a son process, Exit only suspends Suprtool, while Exit Abort actually terminates Suprtool. If you use Exit Abort to terminate Suprtool in a batch job, the MPE Job Control Word is set to a fatal value which normally flushes the remainder of the batch job. In batch, use Exit Suspend instead to quit immediately without setting the JCW to fatal.

Examples

You began to specify a sort, stopped for coffee, and decided to cancel the task on your return:

```
>base store
>get m-customer
>sort name-last; sort name-first
  ... coffee break ...
>exit abort                                  {cancel the sort and terminate}
End Of Program
```

Exit Suspend [ES]

When running Suprtool as a son process (e.g., from Qedit), it would be nice to suspend Suprtool without executing the current task. Exit SUSPEND does this. After returning to Suprtool, all specifications for the current task are still in effect.

Examples

```
:run qedit.pub.robelle
/:run suprtool.pub.robelle
>base store.data
>get d-sales                                {select input}
>sort purch-date                            {start specifying options}
>exit suspend                               {return to Qedit without doing Xeq}
SUPRTOOL.PUB.ROBELLE is still alive. Okay to HOLD ? Y
/...
/:activate suprtool
SUPRTOOL.PUB.ROBELLE
>if sales-total>100000                      {continue specifications}
>output bigtrans
>xeq                                        {execute the current task}
```

Exit Xeq [EX]

Signal the end of command input and the start of an extract operation. After the Suprtool task completes, Suprtool either terminates, or suspends and awakens a father process (i.e., :RUN from within Qedit). Exit Xeq is the default option (i.e., specifying Exit starts execution of the current task).

Examples

```
:run suprtool.pub.robelle
>exit                                       {no input was specified}
No action taken.
End Of Program

:run suprtool.pub.robelle
>input rep23.newprog
>out rep23.program
>xeq                                        {copy one file}
>input rep24.newprog
>out rep24.program
>exit                                       {copy and stop}
```

Notes on Exit Xeq

If you have entered neither sort keys nor an input source, Exit terminates Suprtool without performing any task. If you have defined an input source but without any sort keys, Suprtool does a copy operation prior to stopping. If you have entered only sort keys and have not defined an input source, Suprtool does a sort from a file named Input to a file named Output; Suprtool assumes you have defined Input and Output using :File commands.

Export Command [EXP]

Send commands to STExport.

EXPORT [*STExport-command*]

Switching into STExport

If you enter Export without parameters, Suprtool executes Stexport.Pub.Robelle. STExport then prompts you for commands, just as it does when you run STExport from MPE. When you exit STExport, you are returned to Suprtool.

```
>export                                    {invoke STExport}
STExport/Copyright Robelle Solutions Technology Inc. 1995-2001
$input sdfile
$columns fixed                             {STExport prompts for commands}
$output pcfile
$exit                                      {do task and return to Suprtool}
```

Passing Commands to STExport

You can send commands directly to STExport by specifying the STExport command as part of the Suprtool Export command. STExport executes the command and returns to Suprtool.

```
>export input sdfile
>export columns fixed                     {send "columns fixed" to STExport}
>export output pcfile
>export exit                              {exit from STExport, not Suprtool}
```

Notes

You can use the Suprtool2 interface and the Export command to "call" STExport. See the Suprcall User Manual for a complete example of how to use the interface to pass commands to STExport.

Extract Command [EXT]

Assembles output records by stringing together fields extracted from input records. There can be up to 255 extracted fields, and the same field may be extracted more than once. Constant values may be used instead of the value of the field from the Input record.

EXTRACT *field* [(*subscript*)] [=*value* |=*field2*] [,...]

EXTRACT *field1* [(*subscript1*)] \ *field2* [(*subscript2*)]

EXTRACT *target-field* [=*expression*]

(Default: *subscript*=entire field)

Field Parameter

Each extracted *field* must be an IMAGE field name, an Allbase column, a field in an SD file, or a Defined field. If the field requires an Item definition, then the Item command must precede the Extract command. Extract specifies a rearrangement of the Input data fields to produce the Output data record; there is no data conversion, unless "output xxx, ascii" is specified.

The *subscript* parameter is valid only for compound items. The total item is extracted if it is compound and no *subscript* is specified.

```
>extract account                    {extract the key value and}
>extract rating                     {   one other field}
>output out1                        {output file has two fields}
```

Cumulative Extracts

The Extract command is cumulative. If two Extracts are specified in one run, all fields of the two Extract commands are used.

```
>extract status,balance,account,purchased
```

is equivalent to

```
>extract status,balance
>extract account,purchased
```

Constants

Extracting Constants

The *value* part of the Extract command is used to place a constant in each record of the output file. In this case, the *field* defines the type and length that the *value* occupies. The *value* portion must match the type of the *field*. String values will be extended with blanks to fill the entire field. If the input data does not have a *field* of the correct size and type, you can create one using the Define command.

```
>ext account                        {key value}
>ext rating=0                       {place the value "0" in the "rating" field}
>output out2
```

The total number of bytes that you can extract for all constants is 5,100 bytes for MPE/iX and HP-UX, and 1,275 bytes for MPE/V.

Packed and Display Constants

When extracting non-negative packed and display constants, Suprtool extracts them as unsigned unless you use a leading plus sign. For the value zero, you can use a leading plus or minus sign to get a positive or negative 0.

```
>def field,1,6,packed
>ext field = 1                          {unsigned 1}
>ext field = +1                         {signed 1}
>ext field = +0                         {positive 0}
>ext field = 0                          {unsigned 0}
>ext field = -0                         {negative 0}
```

Decimal Places

If a field has implied decimal places, Suprtool scales the input values according to the number of decimal places. For example,

```
>item tax  ,decimal,2                  {two implied decimal pts.}
>item total,decimal,2                  {same}
>extract tax   = 1.02                  {specified decimal pts.}
>extract total = 100                   {value stored as 10000}
>output out3
```

Blank Fill

If you want to create an output record that consists of fifty blanks followed by the customer name, use:

```
>define filler,1,50
>ext filler=" "
>ext name
```

String Constants

You can specify a string constant without referring to a field. For example, to leave a space between fields, you must do the following:

```
>extract account," ",rating
>output *,ascii
```

Suprtool uses the length of the string to determine the size of the field. The following example extracts the same fields as the previous example, but each output record identifies the field:

```
>extract "Customer Account=",account," "
>extract "Credit Rating=",rating
>output *,ascii
```

The output would look like:

```
Customer Account=04598921 Credit Rating=      5000
Customer Account=44657844 Credit Rating=     20000
Customer Account=98753198 Credit Rating=      3000
```

The spaces after "Credit Rating=", before the rating value, is due to the numeric field Rating being extracted with blanks for its leading zeros. This is the result of the Ascii option of the Output command.

Repeated Fields

If the *field* is an IMAGE repeated field (e.g., 10J1), the Extract command places the value **in each of the repeated fields when you do not specify the *subscript***. If you specify a *subscript*, only that one repeated field will have the new constant *value*.

In this example, Address is a repeated field (2X20). We wish to extract the data as it exists in the input record rather than forcing it to a constant value.

```
>ext account                    {extract key value}
>ext address                    {take both of the repeated fields}
```

In the next example, we assume that the Balance field is a repeated field (12J2). We wish to make each of the 12 repeated fields in the output record equal to 100:

```
>extract name
>extract balance=100
```

If we only wanted to extract the sixth field of BALANCE and set it to 100 we would do the following:

```
>extract name
>extract balance(6)=100
```

Character Constants

Use the ^-character to specify any ASCII character. The number (the actual ASCII value), or letter (^A means control A), must follow immediately after the ^-character. Suprtool treats character constants as strings. When you extract the constant to a field longer than one byte, Suprtool pads it with spaces.

```
>define field,1,1              {byte field}
>ext field = ^0                {binary zero}
>ext field = ^G                {Control-G (bell)}
>ext field = ^27               {escape}
>ext field = ^252              {Roman-8 box}
>ext field = ^186              {Euro currency symbol €}
```

You can also extract the constant directly without referring to a defined field. This always produces a one-byte constant with no blank padding.

```
>ext ^0                        {binary zero}
>ext ^13,^10                   {Carriage Return, Line Feed}
>ext ^M,^J                     {CR, LF again}
>ext ^27,"&dB"                 {escape sequence}
```

Dates

Extracting Today's Date

To extract today's date, use the following:

```
>item    field,date,ccyymmdd   {identify date format of field}
>extract field=$today          {today's date}
>extract field=$today(-1)      {yesterday's date}
>extract field=$today(+1)      {tomorrow's date}
```

Use the Item command to qualify the field as a date. Suprtool uses the date format to determine the output format of the date. The $today function accepts one optional argument which is the number of days before or after today. The maximum number of days in either direction is 9999.

Oracle dates include both the date and the time. If you extract an Oracle date using $today, the time is always 00:00 (i.e., midnight).

Extracting Relative Dates

The Extract command provides the same relative date features as the If command (see "Date Selection" on page 202 for a complete description of the options of

$date). You must first use the Item command to identify the field name as a date. Suprtool uses the field type and length along with the date format to determine the output format of the date. Note that the three parts of $DATE are always specified in (year/month/day) order, regardless of the date format of the field.

```
>item    field,date,mmddyy
>extract field=$date(*/*/*)            {today's date}
>extract field=$date(*/*-1/01)         {start of last month}
>extract field=$date(*/*-1/last)       {end of last month}
```

Oracle dates include both the date and the time. If you extract an Oracle date using $date, the time is always 00:00 (i.e., midnight).

$Stddate

Similar to the If command, the Extract command is also capable of utilizing the $stddate function. This will allow for conversion of any of the supported Suprtool date formats to be converted to a date in the ccyymmdd date format in a double integer container.

For example,

```
>get sales-detail
>def new-ship-date,1,4,double
>item ship-date,date,mmddyyyy
>ext order-no / sales-amount
>ext new-ship-date = $stddate(ship-date)
>out salesinfo,link
>xeq
```

Invalid Dates

Because the $stddate must have a valid date in order to properly convert the date to the ccyymmdd format, a value of 0 will be returned for any invalid dates. An invalid date is any number of a particular date format whose date equivalent cannot be found on the calendar.

This means that if you attempt to extract use the $stddate function against a value that is not a valid date then the extracted value will be 0.

$Days

As with the $stddate function, the $days function is also available to the Extract command. You can convert any supported date to a Julian Day number in the following manner:

```
>in ordfile
>def ship-days,1,4,double
>def order-days,1,4,double
>def delay,1,4,double
>ext order-no
>ext ship-days=$days(ship-date)
>ext order-days=$days(order-date)
>ext delay=$days(ship-date)-$days(order-date)
>out neword,link
>xeq
IN=15, OUT=15.  CPU-Sec=1.  Wall-Sec=1.
```

Invalid Dates

If an invalid date is encountered, the extracted value will be zero. Therefore, in the example above, if the order has not yet been shipped (ship-date does not contain a valid date) the resulting delay value will be negative.

Add and Subtract Dates

With the $days function, you can generate a date that is *n* days before or after any date. In the following example, we show you how to get the previous day's date.

```
>input YOURFILE
>def origdate,1,8
>def yesterday,1,8
>item origdate,date,yyyymmdd
>item newdate,date,yyyymmdd
>ext origdate
>ext yesterday = $stddate($days(origdate) - 1)          {or +7 for next week}
>out tmpfile,link
>xeq
```

Sample output:

```
ORIGDATE YESTERDAY

19990101 19981231
19991231 19991230
19990301 19990228
```

Date Limits

The $date function in Suprtool can generate dates between the years 1583 and 2583. Some date formats have limits based on their particular format, such as 2027 for a Calendar date and 2259 for the aammdd aamm, mmddaa, ddmmaa dates.

Range of Fields

Extracting a Range of Fields

You can specify a range of fields to extract using the following:

Extract Field1 \ Field4

Extract $first \ $last

Extract $All {means $first \ $last}

This feature only works with IMAGE fields, input files "equated" to an IMAGE dataset and for self-describing files. If you specify a range, Suprtool extracts all 4 of the field names between field1 and field4 inclusive. When specifying a range of a self-describing file that has been equated to an IMAGE dataset, the IMAGE dataset definition takes precedence.

```
>get d-sales                                  {input is from a dataset}
>ext product-no\sales-qty
>out dsales
>xeq
```

is exactly the same as:

```
>get d-sales                              {input is from a dataset}
>ext product-no                          {first field in the range}
>ext product-price
>ext purch-date
>ext sales-qty                           {last field in the range}
>out dsales
>xeq
```

Extracting a Range of Subscripted Fields

Suprtool accepts a subscript on either field in a range. You can even use this feature to extract a range from a single field. For example, if sales-amt is a 12J2 field:

```
>get d-sales                              {input is from a dataset}
>ext sales-amt(4)\sales-amt(6)
>out dsales
>xeq
```

is equivalent to:

```
>get d-sales                              {input is from a dataset}
>ext sales-amt(4)                        {first subscripted field}
>ext sales-amt(5)                        {intermediate subfield}
>ext sales-amt(6)                        {last subscripted field}
>out dsales
>xeq
```

Alternate Syntax for Extracting a Range

Suprtool accepts a slash "/" in place of the backslash "\" to specify a range. Use the slash with care, because it is a valid character in field names. For example,

```
>extract a/b
```

would produce the error message:

```
Error:  Item "A/B" not in the database...
```

To use a slash in an extract range, surround it with spaces:

```
>extract a / b
```

You can also use $first, $last or all in Extracting a range. The intention is to make your scripts more easily maintained. If you had a script that you wanted to put a sequence number at the beginning and then extract the rest of the dataset you could specify the starting field and the ending field. For example:

```
base orddb
get customers
def seq-no,1,4,double
ext seq-no=$counter
ext order-no / pst-code
out newfile,link
xeq
```

If you added any fields to the beginning or end of the dataset and you used the specific fieldnames you would have to re-write the script. However, you can use the $all feature in extract range:

```
base orddb
get customers
def seq-no,1,4,double
ext seq-no=$counter
ext $all
out newfile,link
xeq
```

You can also write the script using $first / $last as your preference, but $first and $last are also useful if you need to add data into the middle of the fields you extract:

```
base orddb
get customers
def seq-no,1,4,double
ext $first / zip
ext seq-no=$counter
ext tax-code / $last
out newfile,link
xeq
```

Numeric Expressions

You can specify arithmetic expressions for any numeric data-type in the Extract command. Arithmetic expressions involve the operators +, -, *, / and mod. Extract arithmetic expressions work exactly as If command arithmetic expressions. To extract an expression, use this syntax,

EXTRACT *target-field = expression*

Target-Field

The *target-field* determines the byte-length, data-type, and repeat-count for the expression. The expression is extracted during the output phase and cannot be used by other Suprtool commands that accept fields (e.g., sort). To avoid confusion, it is best to define a new field name for the *target-field* instead of using an existing field name.

Examples

```
>extract budget99 = actual98 + 1000
>extract total = cost * qty
>extract day = ccyymmdd-date mod 100
```

In the following example, the field `total` is used twice. In the first case, it is used to tell Suprtool how to format the arithmetic expression. In the second case, it is used in the sort command. Warning: In this example, the output file is sorted by the value of `total` as it appears in the input record. It is not be sorted by `cost * qty`.

```
>extract total = cost * qty
>sort    total                              {sort by input total}
```

Restrictions

You can only use one expression in each Extract command, and the expression must be the last item. If you want to extract several expressions or more fields after an expression, you need to use several Extract commands.

Incorrect

```
>extract name, i=sales + tips, c=cost + expense, dept
```

Correct

```
>extract name, i=sales + commission
>extract c=cost + expense
>extract dept
```

Constants vs. Expressions

If you have an arithmetic expression that starts with a constant, Suprtool assumes that you are attempting to extract a single constant value and not an arithmetic expression. To specify an arithmetic expression that starts with a constant, surround the expression with parentheses. For example,

Incorrect

```
>extract c = 6000 - cost
Error:  Missing comma or invalid arithmetic expression
```

Correct

```
>extract c = (6000 - cost)
```

Numeric Truncation

The accuracy of arithmetic computations is limited to approximately sixteen digits. Suprtool may truncate four-word integers (quad), or large packed-decimal numbers, or display numbers when they are converted to floating-point. Suprtool does not produce any error or warning in this case.

Division by Zero

Suprtool reports an error in the input record number if an arithmetic computation results in division by zero. Use Set Ignore On to force Suprtool to ignore division by zero errors. With Set Ignore On, the result of division by zero is zero.

The speed of a task decreases when you ask Suprtool to ignore many division by zero errors. It is better to check for zero in the If expression before using it in division.

```
>if qty <> 0                                {to avoid division by zero}
>extract average = total / qty
```

$Abs function

Suprtool supports an $abs function, which returns the absolute value of a number. For example, if a field called Credit contains the value -547.83, the $abs function returns 547.83.

This function will work on a field or even on an expression such as:

```
>def newcredit,1,4,double
>ext newcredit = $abs(credit / 100 * 1.07)
```

This function will also work in the If command:

```
>if $abs(credit / 100 * 1.07) > 500.00
```

$Truncate function

Suprtool supports a $truncate function which returns the number to the left of a decimal place. For example, if the field stddev contains the value 547.83, the $truncate function will return 547. Note that there is no rounding.

This function will work on fields and expressions. For example,

```
>def newdev,1,4,double
>ext newdev = $truncate(stddev / 100 * 1.07)
```

This function will also work in the If command:

```
>if $truncate(stddev / 100 * 1.07) > 200
```

$SubTotal Function

Suprtool has the ability to keep a running subtotal for any numeric field based on a given sort key. The target data must be a packed field with 28 digits, in order to avoid overflow issues.

A sample use of the $subtotal function could be:

```
  >def mytotal,1,14,packed
  >get orders
  >sort order-number
  >ext order-number
  >ext part-number
  >ext description
  >ext sales-amount
  >ext mytotal = $subtotal(sales-amount,order-number)
  >out sales,link
  >xeq
```

This would result in a file containing a running subtotal in the field mytotal for a given order-number. You could then generate a simple report with the simple Suprtool commands:

```
  >in sales
  >list standard
  >xeq
```

The basic syntax for the $subtotal function in the extract command is:

```
  extract targetfield = $subtotal(field,sort-field)
```

You must specify the sort command before referencing the sort-field in the $subtotal function. You can subtotal up to ten fields per pass and the $subtotal function is also available in the if command, however, by nature it is of limited use.

$Total Function

Suprtool has the ability to keep a running total for any numeric field. The target data must be a packed field with 28 digits, in order to help avoid overflow issues. A sample use of the total function could be:

```
  >def mytotal,1,14,packed
  >get orders
  >ext mytotal = $total(sales-amount)
  >xeq
```

You can total up to ten fields per pass and the $total function is also available in the if command, however, is of limited use.

$Counter Function

For years Suprtool has had the ability to output a record number to an output file with the num option of the output command:

```
>in mysdfile
>out myfile,num,data
```

The above could would generate an output file called myfile, however, you would lose the SD information and you can only put the number at the beginning or the end of the data. Suprtool now has a counter function that allows you to place a $counter at any spot as well as preserve the SD information.

```
>in mysdfile
>def mycount,1,4,double
>ext field1
>ext field2
>ext mycount=$counter
>out myfile,link
>xeq
```

The file myfile will be self-describing and contain the fields field1, field2 and mycount. The field mycount is defined as a double integer, since this is the only field type that the $counter function can use. Each record will have a unique ascending number starting with one.

String Expressions

You can combine byte-type fields together and use the built-in string functions to create new fields out of existing ones. This can reduce the number of tasks required to provide a solution. String expressions may involve the + operator and $upper, $lower, $trim, $ltrim or $rtrim. To extract a string expression, use this syntax:

EXTRACT *target-field = expression*

Target-Field

The *target-field* determines the byte-length for the expression. The data-type must be Byte or Char. The expression is extracted during the output phase and cannot be used by other Suprtool commands that accept fields (e.g., Sort).

Examples

```
>extract id-no = warehouse-no + bin-no
>extract full-name = first-name + last-name
```

Constants vs. Expressions

If you have an string expression that starts with a string, Suprtool assumes that you are attempting to extract a single string value and not an string expression. To specify a string expression that starts with a constant, surround the expression with parentheses. For example,

Incorrect

```
>extract name = " " + product-desc
Error:  Missing comma or invalid arithmetic expression
```

Correct

```
>extract name = (" " + product-desc)
```

Variable Length Strings

String expressions use variable-length strings. Suprtool keeps track of the length of every string, and all operations are done using the actual string length. For fields, the

length of the string is the length of the field. If you do not want to retain all the spaces in a field, use one of the built-in trimming functions.

String constants are created with the exact length of the constant. For example, the string "abc" is three characters long and the string "a" is one.

When assigning the string expression to the target field, Suprtool pads the final string value with spaces to fill out the target field. String expressions longer than the target field generate an error.

```
>in testfile
>def a,1,10,byte
>ext a="I'm too long for this container"

Error:  String is too long for the specified item
```

String Truncation

Suprtool produces an error if the string expression is longer than the target field. You cannot override this error with Set Ignore On. To help avoid the error, you may want to trim the expression of trailing spaces before assigning it to the target field. For example,

```
>extract new-field = $trim(a + b + c)
```

Upshifting Strings ($Upper)

Use the built-in function $upper to upshift all the characters of a string expression into uppercase characters. This function can be used to upshift a single field, a complicated string expression, or any subpart of an expression. Both ASCII and Roman-8 characters are upshifted by $upper. For example,

```
>extract city-up = $upper(city)
>extract full-name = $upper(first + last)
```

Downshifting Strings ($Lower)

If you want to downshift all characters of a string expression to lowercase, use the built-in function $lower. This function can be used to downshift a single field, a complicated string expression, or any subpart of an expression. Both ASCII and Roman-8 characters are downshifted by $lower. For example,

```
>extract city-lower-case = $lower(city)
>extract city-state = $lower(city + state)
```

Trimming Spaces Using $Trim, $LTrim, $RTrim

Use one of three built-in string functions to remove leading or trailing spaces from a string expression. The three functions are:

$Trim:	Remove leading and trailing spaces from the string expression.
$LTrim:	Remove leading spaces.
$RTrim:	Remove trailing spaces.

Splitting Variable Length Strings

Suprtool can extract portions of a byte field based on the occurrence of certain characters.

Consider the following Data:

```
Armstrong/ Neil/ Patrick
Green/ Bob/ Miller
Edwards/ Janine/
Armstrong/Arthur/Derek
```

The $split function can extract each token into separate fields. The syntax for the $split function is:

```
  $split(Field,Start Character,Occurrence,End Character,Occurrence)
```

The following task will $split the data in the wholename into three separate fields. The below task assumes that the file namefile is self-describing and has the field wholename in it:

```
>in namefile
>define lastname,1,30
>define firstname,1,20
>define middlename,1,20
>extract lastname = $split(wholename,first,"/")
>extract firstname=$trim($split(wholename,"/","/"))
>extract middlename=$trim($split(wholename,"/",2," ",2))
>out names,link
>xeq
```

The first extract statement tells Suprtool extract the bytes from the field wholename, starting at the beginning (first keyword), and stopping at the "/" character. The second extract statement tells Suprtool to extract the bytes between the first occurrence of the "/" character to the next occurrence of the "/" character, and then that string is trimmed of spaces as it is nested within the $trim function.

The third and final extract statement tells Suprtool to extract the bytes beginning with the second occurrence of the "/" character to the second occurrence of the space character.

If the target field is not long enough to hold the data Suprtool will abort with an error. You can easily prevent this from happening on blank fields by nesting the $split statement within a $trim or $rtrim function.

First/Last

The $split function also has a Last keyword, whereby you can split the field from a given occurrence of a character to the end of the field. So, in the given example from above the extracting out of the middlename could be coded as such:

```
>extract middlename=$trim($split(wholename,"/",2,last))
```

The above means to extract out all the data from the second occurrence of the "/", to the end of the field and trim all spaces. Naturally as noted above we also have the First keyword, which indicates the start of the field.

Unprintables

You can specify an unprintable character for Suprtool to use as the character to $split on, using the following syntax:

```
  >extract middlename=$split(wholename,^9,2,last)
```

which means that the split would start at the second occurrence of Decimal Nine, or the Tab character. Please note that for specifying unprintable characters you do not put in quotes.

Cleaning your Data

In this day and age of migrations we were looking at issues that customers have run into when importing data into new databases. What came from this investigation where ways to Clean up your data in any given byte type field.

We have added two methods to clean your data, you can use Suprtool to clean an individual byte type field, or STExport to clean all of the byte-type fields for a given file that you are exporting.

Sometimes un-printable or extraneous characters get stored in files or databases that have no business being there. This may be some tab characters in an address field or perhaps and embedded carriage return or line-feed. Suprtool now supports the clean function which will replace individual characters for a given byte field.

There are three things that Suprtool needs to know in order to "clean" a field. Suprtool needs to know which characters it needs to clean, what character it needs to change the "bad" characters to, and also what field does it need to clean.

Clean Command Syntax

You can specify special characters Decimal 0 thru Decimal 31 via the command:

```
Clean special
```

You can also specify a range or characters by using the following syntax:

```
Clean "^0:^31","^240:^255"
```

The Clean command is used to tell Suprtool what characters it needs to look for in a given byte type field. For example:

```
clean "^9","^10","."
```

will tell Suprtool to replace the tab character (Decimal 9), Line Feed (Decimal 10), and a period to whatever the Clean character is set to. The CLean command takes both, decimal notation and the character itself, however, it is probably most convenient to use the Decimal notation for the characters that you wish to clean. The Decimal notation is indicated by the "^" character.

Setting the Clean Character

By default, Suprtool will replace any of the characters specified in the clean command with a space. You can specify what character to use to replace any of the characters that qualify with the following set command:

```
>set CleanChar "."
```

This will set the character to replace any of the qualifying "to be cleaned" characters to be a period.

Cleaning a Field

You call the clean function, the same way you normally use other functions available to if and extract. For example:

```
ext address1=$clean(address1)
```

shows how to clean the field address1. You do not necessarily need to have the target field be the same as the source field.

```
def new-address,1,30
ext new-address=$clean(address1)
```

Cleaning your data

An example of how easy it would be to clean your database of certain "bad" characters in byte-type fields would be as follows:

```
>base mydb,1,;
>get customer
>clean "^9","^10","^0","^7"
>set cleanchar " "
>update
>ext address(1) = $clean(address(1))
>ext address(2) = $clean(address(2))
>ext address(3) = $clean(address(3))
>xeq
```

The above task will look at the three instances of address and replace the tab, linefeed, null and bell characters with a space.

If you want to just remove the characters all you need to do is set the CleanChar in the following manner:

```
>Set CleanChar "<null>"
```

This means that the $clean function will remove the characters specified in the clean command, but not replace with any character, which effectively shifts the text to the left and pad the equivalent amount of spaces at the end.

Extract from a Table

Suprtool has the ability to load data into a table via the Table command, and extract that data out of the table using the Extract command. The Extract command can utilize the $lookup function to return data. The syntax for the $lookup function would look as follows:

```
>extract target = $lookup(table-name,key-field,data-field)
```

The Table name, key-field and data-field are all defined by the Table command, which must be input before the Extract command. A classic example: your boss comes to you with a list of new prices for certain parts and asks you to update the Part-Master dataset.

Just load the new prices into a Table, index by the product number (prodno), then Extract the price field from each record and replace it with a $lookup on the table. Here is the code:

```
>table newprices,prodno,file,bosslist,data(price)
>get part-master
>if $lookup(newprices,prodno)
>update
>extract price = $lookup(newprices,prodno,price)
>xeq
```

We do the If $lookup to select only the parts which have new prices, then do Extract with $lookup to replace the existing price with a new one. The Update command forces a database update on each selected record and must come before the Extract command.

Now let's see how a Table can be used to add additional useful information to a record. Let's say we build this table of Canadian provinces (The file prov-file is

assumed to be a Link, or self-describing, file, created by a previous pass of Suprtool.)

```
>table provtab,prov-code,file,provfile,data(prov-name)
```

At this point the key into the Table is the prov-code item and for each entry in the Table there is one associated prov-name. To append prov-name to each output record, we read the customer dataset, extracting the customer name. We also Define prov-name as a new field and extract it for the output record, but we fill it with a value that is based on the prov-code for each customer entry:

```
>get  customers
>ext  cust-name
>def  full-prov-name,1,30
>ext  full-prov-name=$lookup(provtab,cust-prov-code,prov-name)
>out  somefile
>xeq
```

To update a dataset, you do the same commands, but you insert an Update command prior to the Extract from a Table. Below is an example that shows how to update an IMAGE record using data values from a Table.

Let's assume that we have new unit cost information for each product:

```
>form newcosts
 File: NEWCOSTS.NEIL.GREEN     (SD Version B.00.00)
 Entry:                      Offset
    PRODNO               Z8      1
    UNIT-COST            P8      9
 Limit: 13  EOF: 13  Entry Length: 12  Blocking: 64
```

We load a table with the product number key value (prod-no)and the new unit cost data value (unit-cost):

```
  >table prodcost-table,prodno,file,newcosts,data(unit-cost)
```

We can then select that unit-cost field from the prodcost-table using the Extract command:

```
  >extract unit-cost = $lookup(prodcost-table,prodno,unit-cost)
```

Here is the entire task, keeping in mind that Update must be specified before the Extract command.

```
  >base store.suprtpis
  Database password [;]?
  >get d-inventory
  >table prodtable,prodno,file,newcosts,data(unit-cost)
  >update
  >if $lookup(prodtable,prodno)
  >extract unit-cost = $lookup(prodtable,prodno,unit-cost)
  >xeq
```

If you did not specify the If $lookup, then records that did not qualify under the $lookup function in the extract field, will result in zeroes for any numeric field and spaces for any byte type fields.

Data Conversion

You can convert numeric fields from one data-type to another. Any nonbyte field type is considered to be numeric. You can also lengthen or truncate character fields. The general syntax for doing conversions is:

EXTRACT *target-field = source-field*

Target-Field

The *target-field* determines the byte-length, data-type, and repeat-count for the expression. The expression is extracted during the output phase and cannot be used by other Suprtool commands that accept fields (e.g., Sort). To avoid confusion, it is best to define a new field name for the *target-field* instead of using an existing field name.

The following example shows defining a new *target-field* as a double integer. The Extract command *target-field* then takes the definition from the Define command and extracts data from the *source-field* (display-field).

```
>in oldfile
>def salesqty,1,4,double
>ext order-no / order-date
>ext salesqty = display-field
```

Packed and Display Fields

When the target of an extract conversion is a packed- or display-type field, Suprtool always converts positive values to a neutral packed- or display-value. To ensure that expressions with positive values have a positive result, use the $signed function:

```
>extract packed-field = $signed(int-field)
>extract display-field = $signed(dbl-field / 10)
```

Truncation errors can occur when Suprtool converts from nonfloating-point to floating-point. See the discussion under *Numeric Truncation* above.

Byte Fields

Use the Extract command to shorten or lengthen byte-type fields. This feature is most useful when updating one field in an IMAGE/SQL dataset with another field.

```
>get     d-sales
>define  part-warehouse-no,part-no[5],4
>extract warehouse-no = part-warehouse-no
>update
>xeq
```

If the *target-field* is longer than the *source-field*, Suprtool fills the trailing space in the *target-field* with spaces.

Byte to Numeric Conversion

Suprtool cannot explicitly convert from a byte field to a numeric field such as a double integer. The Extract command, however, does allow conversion from a display field to a double integer (or any other numeric field).

You can define a byte field to be a display field if all of the characters in the field contain a number. For example, if you have a six-character byte field that looks like this:

```
012345
```

you can define it in the following manner:

```
>def display-field,1,6,display
```

This field can then be converted to any of the other numeric types that Suprtool supports.

If the field is six characters and contains blanks, decimal, currency or a sign symbol in the data then you can utilize the $number function.

$Number Function

Suprtool now has the ability to accept free-form "numbers" as display data types. This means numbers in the form:

```
 1234.45-
 -12345
 -123.2134
  12343
 $123.45
```

can now be accepted and converted to any other numeric data type. Consider the following data:

```
  Item-number    New-Price
   12345          +123.45
   34563          + 27.5
   21312          +  1.545
```

Suprtool can now read and convert the data in New-Price using the number function. Let's say we want New-Price to be a double integer and currently occupies eight bytes starting in position six. Here is the task you would use to convert the New-Price free-format number into a double integer.

```
  >in mynums
  >def item-number,1,5,byte
  >def new-price-ascii,6,8,display
  >def new-price,1,4,double
  >item new-price-ascii,dec,2
  >item new-price,dec,2
  >ext item-number
  >ext new-price=$number(new-price-ascii)
  >out somefile,link
  >xeq
```

The $number function take the free-format number and make it a valid display number. It will determine the decimal, sign and add leading zeroes. It will round the number to the defined number of decimal places.

In the case of 1.545 number, Suprtool will round the value to be 1.55, since the given number of decimal places is two and the following value is five or greater. If you have a whole number such as 54, with no decimal point the value becomes 54.00.

Suprtool will not accept data that has:

```
   More than one sign.
   More than one decimal place.
   Spaces in between numbers.
   Signs that are in between numbers.
   Characters that are not over punch characters.
   Fields that when edited do not fit in the defined space for the
display field.
```

You can control the character that defines the currency, thousand and decimal symbol for other currencies and formats using the following commands:

```
   >set decimalsymbol "."
   >set thousandsymbol ","
   >set currencysymbol "$"
```

Suprtool in the above case will strip the currency and thousand symbols and use the decimal symbol to determine the number of decimal places. You can set these characters to any values you want but the defaults for each are used in the above set commands. The decimal and thousand symbols are only single characters. The currency symbol allows for four characters.

Numeric to Byte Conversion

Suprtool has several ways to convert binary numbers (e.g., J4, I2, P8) into human-readable ASCII form. You can use STExport or Suprtool's Output,Ascii or Output,Display commands.

If you want to convert only some of your numeric fields, you can use Suprtool's numeric conversion to convert binary fields to display fields. For example, here is a conversion of a J4 field to an Z18 field:

```
define mynumber 1,18,display
get dataset
extract some-fields...
extract mynumber = binary-number
output filename
xeq
```

You can also use the $Edit function to format and directly convert to byte format.

$Edit Function

Suprtool can format fields using edit-mask features similar to edit-mask features of Cobol. Suprtool employs two distinct types of edit-masks: one for byte type fields and the other for numeric fields.

The type of mask utilized depends on the source type of the field. If the source field is numeric, then the numeric edit-mask logic is applied, if the source field is byte type, then the byte edit-mask logic and characters apply.

The target field must always be a byte type field.

Placeholders and Format Characters

An edit-mask consists of "placeholder" characters, such as "9" for a numeric column, and "format" characters, such as "." for the decimal place. Sometimes an edit-mask character acts as both a placeholder and a format character, such as the "$" in floating dollar signs.

Byte-Type Formatting

For Byte type fields there are two placeholder characters. These are:

X ~ place the data in the matching column for the X in the edit-mask

Z ~ place the data in the matching column unless the data is a zero; if the data is a zero, then replace with a space

The format characters are as follows:

B (space) / (slash) , (comma) . (period) + (plus) - (minus) * (asterisk)

and a space. Please note that you can denote a space using two methods, either by putting a "B" in the mask or a space itself. For example, suppose you have data that is in ccyymmdd format in an X8 field. Here is how you would use a "xxxx/xx/xx" mask to format the data:

```
>in mydate
>form
   File: MYDATE.TEST.NEIL (SD Version B.00.00)
      Entry:              Offset
         A       X8       1  <CCYYMMDD>
   Limit: 10000  EOF: 2  Entry Length: 8
>def formatdate,1,10
>ext formatdate=$edit(a,"xxxx/xx/xx")
>list
>xeq
>IN MYDATE.NEIL.GREEN (0) >;OUT $NULL (0)
FORMATDATE   = 2003/09/24

>IN MYDATE.NEIL.GREEN (1) >;OUT $NULL (1)
FORMATDATE   = 2003/09/24
```

As you see in the example above, the placeholder character is the "x" and the "/" is the format character. You insert a space either by specifying a "B" or by putting an actual space character in the edit-mask. An example of inserting a space might be the formatting of Canadian postal codes (e.g., V3R 7K1):

```
>in postal
>form
   File: POSTAL.NEIL.GREEN
      Entry:                      Offset
         POSTAL-CODE           X6      1
   Limit: 10000  EOF: 2  Entry Length: 6
>def post1,1,7,byte
>def post2,1,7,byte
>ext post1=$edit(postal-code,"xxx xxx")
>ext post2=$edit(postal-code,"xxxbxxx")
>list
>xeq

>IN POSTAL.NEIL.GREEN (0) >OUT $NULL (0)
POST1     = L2H 1L2     POST2     = L2H 1L2

>IN POSTAL.NEIL.GREEN (1) >OUT $NULL (1)
POST1     = L2H 1L2     POST2     = L2H 1L2
```

Z-placeholder for byte-fields

The Z-placeholder character works differently for byte-fields than for numeric fields. For byte type fields, if the Z placeholder and the corresponding data is "0", then the zero is suppressed, regardless of the position. This is primarily for suppression of zeroes in byte type date fields:

```
ext a=$edit(date-field,"xxxx/zx/zx")
```

The above edit mask would then edit a byte type date of 20031005, to be:

```
2003/10/ 5
```

Overflow and limits

An edit mask is limited to 32 characters in total for both numeric and byte type fields. If data overflows the edit-mask, by default Suprtool will fill that field with asterisks. There is an option to have Suprtool stop when it encounters a formatting overflow:

```
>set editstoperror on
```

will force Suprtool to stop if there is data left over after applying the edit-mask. With byte-type fields, leading spaces do not cause overflow. Therefore, if your data consists of:

```
"    L2H1L2"
```

and your edit mask is:

```
"xxxBxxx"
```

It is not an overflow since there are only spaces to the left of the "L". If the data was:

```
"   JL2H1L2"
```

an overflow exception would occur.

Numeric field edit-masks

Our edit-masks for numeric fields are patterned after those in COBOL. We provide four placeholder characters, each with a slightly different effect:

"9" - insert a digit from 0 to 9 in this position

"$" - if you specify more than one dollar sign, you get a floating dollar sign. This means that there can be as many numeric positions as there are dollar signs, but if some positions are not needed because the value is small, the $ floats to the right next to the first digit and the preceding positions are blank.

"*" - if there are enough digits in the value, the * position is replaced by a numeric digit; if not, an asterisk is printed. Leading asterisks are often used for check writing, so that no one can insert a different value.

"z" - insert a numeric digit at this position; if the rest of the data to the left is a zero then a space will be placed at this position. For example:

```
>ext a=$edit(int-field,"$$,$$$.99-")
>ext b=$edit(int-field,"99,999.99-")
>ext c=$edit(int-field,"cr99999.99")
>ext d=$edit(int-field,"-$9999.99")
>ext e=$edit(int-field,"**,***.99+")
>ext f=$edit(int-field,"zz,zzz.99+")
>list
>xeq
>IN FILE1SD.NEIL.GREEN (0) >OUT $NULL (0)
A      =    $11.11-     B   = 00,011.11-
C      = CR00011.11     D   =  -$0011.11
E      = ****11.11-     F   =     11.11-

>IN FILE1SD.NEIL.GREEN (1) >OUT $NULL (1)
A      =    $22.22-     B   = 00,022.22-
C      = CR00022.22     D   =  -$0022.22
E      = ****22.22-     F   =     22.22-
```

Signs

As shown in the example above, there are also numerous format characters for numeric edits, including four ways to specify the sign. You can specify a sign, with +, -, or the typical accounting specification of "CR" and "DB". You will note in the example above that the "cr" in the mask was up-shifted to be "CR". This is because the entire mask is up-shifted as the mask is being parsed.

You can specify more than one sign in a numeric field edit, although Suprtool will give you a warning that having two sign edit-mask characters does not really make sense. Cobol gives a Questionable warning when compiling an edit-mask with two sign characters. Suprtool, will apply the sign in both places.

Keep in mind that most data has three states:

1) Postive
2) Negative
3) Neutral

Any neutral data will not display the sign. If you specify a "+" sign in the edit-mask and the data is negative, it will of course display a "-" sign.

Decimal Places

For numeric-type edits, Suprtool attempts to adjust the data according to the number of decimal places in the edit-mask, when compared to the number of decimal places defined in the field.

For example, if the data field has one decimal place, and the edit mask has two decimal places, then the data is adjusted:

Data and Edit mask:

```
  102.3   ZZZZ.99
```

will result in the final data being:

```
  102.30
```

Similarly, if the data has three decimal places and the edit-mask only has two, then the data will be rounded appropriately with the same rules as outlined in the $number function.

You can specify more than one decimal place in an edit-mask. However, Suprtool will print a warning and it will utilize the right-most decimal place for data alignment. The decimal place character is defined by a set command:

```
   >set decimalsymbol "."
```

If you define another character as the decimal symbol, Suprtool will use that character as the point to align the decimals. If you define a decimal symbol that is not an allowed edit-mask character with Set Decimalsymbol, Suprtool will assume that the field has zero decimal places and adjust the data accordingly.

Currency and Dollar signs

Suprtool edit-masks support both fixed and floating dollar signs. Logic for floating dollar-signs will be invoked if more than two dollar signs are defined in the edit-mask.

A floating-dollar edit mask attempts to put the dollar sign at the left most position of the significant data. For example, if you have the following data and edit mask:

```
0001234.54 $$$$$$.$$
```

the data would end up as:

```
   $1234.54
```

Suprtool will not however, put the dollar sign to the right of the decimal place. If you had the same edit mask and the data was, .09, the data would end up being formatted as:

```
       $.09
```

Similarly, the $edit function will attempt to place the dollar sign correctly in most cases. For example, Suprtool will not format data in the form of:

```
   $,123.50
```

Suprtool, does attempt to fixup these cases and would format the data in the following manner:

```
    $123.50
```

Overflow and floating dollar

If the number of digits in the data is equal to the number of placeholder dollar signs, then the dollar sign is dropped and not added to the edited field.

```
12345.50 $$$$$.99
```

would result in:

```
12345.50
```

Set CurrencySymbol

If Set CurrencySymbol is not equal to "$", then after the formatting has been applied, whatever symbol(s) are defined within the set command, are used to replace the "$" symbol in the data. For example, if you have the Currency symbol set as "CDN".

```
  >set currencysymbol "CDN"
```

Suprtool will replace the "$" after the edit-mask has been applied with CDN, provided there is room to the left of the dollar-sign. It is recommended that if you are using multiple characters for the dollar symbol that you leave enough characters to the left of the symbol.

For example, if the CurrencySymbol is defined as CDN, then you should leave two spaces to the left of a fixed dollar sign definition. If there is not enough room, to put in the currency symbol, then the dollar symbol is blank.

Overflow and limits

An edit mask is limited to 32 characters in total for both numeric and byte type fields. If data overflows the edit-mask, by default Suprtool will fill that field with asterisks. There is an option to have Suprtool stop when it encounters a formatting overflow:

```
  >set editstoperror on
```

will force Suprtool to stop if there is data left over to place when applying the edit-mask. With numeric-type fields, leading zeroes do not cause overflow.

Restrictions

You can only use one expression in each Extract command, and the expression must be the last item. If you want to extract several expressions or more fields after an expression, you need to use several Extract commands.

Incorrect

```
>extract name, i=sales + tips, c=cost, dept
```

Correct

```
>extract name, i=sales + commission
>extract c=cost
>extract dept
```

Extracting Bits

The Extract command can be used to define individual bits from one data item as separate fields.

```
>def order-shipped,1,2,int
>def order-paid   ,1,2,int
>ext order-shipped=status-field.(0:1)
>ext order-paid=status-field.(1:1)
```

This makes it easier to check the status of certain bits within a given field.

EBCDIC Conversions

Use the $etoa or $atoe functions to convert specific fields from EBCDIC to ASCII or vice versa. Each of these functions accepts a single parameter that is a byte-type field:

Extract $ETOA(char-field)

Extract $ATOE(char-field)

There are several restrictions on the $etoa and $atoe functions:

- They do not work with either the ASCII or PRN output options.
- You cannot extract an EBCDIC constant. The following example would produce an error message:
  `>extract $etoa(char-field) = 'abcdef'`
- You cannot extract a range of fields using $etoa or $atoe.

Notes

The Extract command is valid only with

- Output xxx,data
- Output xxx,data,num
- Output xxx,query
- Output xxx,link
- Output xxx,ascii
- Output xxx,prn

The Extract occurs logically after the sort phase, if any, but prior to the final Output, Put, or List. An If command can refer to fields of the input record that are not included in the extracted output record. The sort keys can be fields that are not among those extracted.

If the extracted record length is shorter than the input record length, Suprtool attempts to speed up sorts by doing the extract before sorting. Suprtool can only do the extract before sorting if the output option is DATA (the default), QUERY, or LINK, and all of the sort keys are included in the Extracted fields.

One advantage of not using the Extract command is that the output file from Suprtool has *exactly* the same format as the input dataset which created the file. You can then use the *=setname* option of the Input command to define all of the fields in

the output file. Even if you change your database structure, many of your job streams that use Suprtool and the =*setname* option will not have to be changed.

Form Command [F]

The Form command displays information about the items and datasets in a database or the fields in a self-describing file. The Form command is similar to the Form command of QUERY.

FORM [SETS | ITEMS | PATHS | *dataset* | *data-item* | *filename*]

(Default: depends)

The Form command displays the structure of a database, dataset, table or self-describing file. The Form command is also available in Dbedit. The Query All option is not supported.

When showing the form of an IMAGE dataset or a self-describing file, Suprtool shows the byte offset of each field after the subcount, type, and sublength. The first field always appears at offset one. If you have specified a date format or the number of implied decimal places with the Item command, these attributes appear as part of the form listing.

Example

```
>base store,5
>item last-ship-date, date   , yymmdd
>item unit-cost     , decimal, 2
>form d-inventory

D-INVENTORY      Detail                  Set# 4
   Entry:                      Offset
      BIN-NO                 J1      1
      LAST-SHIP-DATE         J2      3                    <<YYMMDD>>
      ON-HAND-QTY            J2      7
      PRODUCT-NO             Z8     11  (M-PRODUCT)
      SUPPLIER-NO            Z8     19  (!M-SUPPLIER)
      UNIT-COST              P8     27                    << .2  >>
Capacity: 330 (22), 330, 66, 550  Entries: 13  Highwater: 13  Bytes: 30
```

Database Name

For IMAGE databases, Suprtool attempts to show the fully qualified database name. It cannot do this with a remote database, or if the database was opened in mode-3, or if Set Privmode was Off when the Base command was specified. If the database has MPE/iX third-party indexing, the third-party product name and version number are also shown.

```
>ba store
>fo

Database: STORE.DEMO.APPDEV  TPI: SUPERDEX (15015d) 4.0.39

               Set       Item   Capa- Entry  Load    Entry
Sets:          Num Type  Count  city  Count  Factor  Length  B/F
   M-CUSTOMER  1   M     9      211   20     9  %    55      7
   M-PRODUCT   2   M     3      307   13     4  %    24      12
   M-SUPPLIER  3   M     6      211   3      1  %    49      8
   D-INVENTORY 4   DDX   6      550   13     2  %    15      22
   D-SALES     5   D     8      602   0      0  %    19      14
```

Suprtool shows the type of each dataset (e.g., "M" for Manual master, "A" for Automatic master, "D" for Detail). If you have enabled dynamic dataset expansion, Suprtool adds "DX" to the type of the dataset (e.g., "MDX", "DDX").

Detail Datasets

If you request information about a specific detail dataset, Suprtool prints the path information in DBSCHEMA format. The path shows the related master dataset and the sort item-name. When displaying the form of an IMAGE/SQL dataset, Suprtool shows the capacity in a format similar to the one in DBSCHEMA.

```
>form d-inventory

D-INVENTORY      Detail                  Set# 4
   Entry:                     Offset
      BIN-NO                J1      1
      LAST-SHIP-DATE        J2      3                    <<YYMMDD>>
      ON-HAND-QTY           J2      7
      PRODUCT-NO            Z8     11  (M-PRODUCT)
      SUPPLIER-NO           Z8     19  (!M-SUPPLIER)
      UNIT-COST             P8     27                    << .2  >>
Capacity: 330 (22), 330, 66, 550  Entries: 13  Highwater: 13  Bytes: 30
```

The current capacity (e.g., 330) is shown first followed by the blocking factor in parentheses (e.g., 22). If dynamic dataset expansion (a feature of MPE/iX 5.0) has been enabled, the initial capacity (e.g., 330), the increment (e.g., 66), and the maximum capacity (e.g., 550) are shown next. On MPE/iX 5.0 or later, the highwater mark is shown. The record length in bytes appears last.

Master Datasets

The Form Sets command indicates which datasets have MDX enabled. A Form command on these datasets shows details of their expansion setting.

```
>form sets
Database: STORE.MDX.DATA

                Set       Item           Entry  Load   Entry
Sets:           Num Type  Count Capacity Count  Factor Length B/F
   M-CUSTOMER   1   MDX   9     2009     401     20 %   55     7
   M-PRODUCT    2   MDX   3     3012     0        0 %   24     12
   M-SUPPLIER   3   MDX   6     2008     0        0 %   49     8
   D-INVENTORY  4   D     6     462      0        0 %   15     22
   D-SALES      5   D     8     6006     0        0 %   19     14

>form m-customer

Database: STORE.MDX.DATA

  M-CUSTOMER       Master                  Set 1
     Entry:                     Offset
        CITY                  X12     1
        CREDIT-RATING         J2     13
        CUST-ACCOUNT          Z8     17  <<Search Field>>
        CUST-STATUS           X2     25
        NAME-FIRST            X10    27
        NAME-LAST             X16    37
        STATE-CODE            X2     53
        STREET-ADDRESS       2X25    55
        POSTAL-CODE           X6    105
Capacity: 469 (7), 217, 252, 2009  Entries: 401  Bytes: 110
```

As you can see, the master dataset summary information for the Capacity shows the Current Capacity (469), the Blocking Factor (7), the Initial Capacity (217), the Increment (252) and finally the Maximum Capacity (2009). The Increment is the number of entries by which the master dataset increases from the current capacity.

SQL Database

If an Allbase database is open and no input file has been specified, the default Form command shows all of the tables in the database. If a Select command has been specified, the default Form command shows the columns in the Select command. The exact format of the Form command is different for each SQL database.

Self-Describing Files

The Link output option produces an SD file with information about how the file was sorted, what fields are compound, and the date format or the number of implied decimal places for any fields. The Form command shows all of this information:

```
>form custfile
File: custfile                    (SD Version B.00.00)
   Entry:                     Offset
      CHAR-FIELD             X5    1  <<Sort #1 >>
      REPEATED-I1           3I1    6     {compound field}
      DATE-FIELD             J2   12  <<YYYYMMDD>>
      COST-FIELD             J2   16  << .2  >>
Limit: 10000  EOF: 15  Entry Length: 20  Blocking: 64
```

Third-Party Indexing

If the database is enabled for MPE/iX third-party indexing, the Form command shows any third-party index that corresponds to an IMAGE field. When doing a Form *dataset*, each field is checked to see if it is a byte-type or Z-type third-party index. If it is, the comment "<<TPI>>" is shown. Indexes that have a name other than an Image-item name are shown at the end of the form listing. The Form command only shows those indexes that can be referenced by the Chain command.

For example,

```
>form m-customer
  M-CUSTOMER       Master    Set# 1
    Entry:                     Offset
        CITY                 X12     1  <<TPI>>
        CREDIT-RATING        J2     13
        CUST-ACCOUNT         Z8     17  <<Search Field>>
                                        <<TPI>>
        CUST-STATUS          X2     25
        NAME-FIRST           X10    27  <<TPI>>
        NAME-LAST            X16    37  <<TPI>>
        STATE-CODE           X2     53  <<TPI>>
        STREET-ADDRESS      2X25    55  <<TPI>>
        POSTAL-CODE          X6    105
  Capacity: 211 (7)  Entries: 20  Bytes: 110

  Additional Third-Party Indexes:
        SI-LAST-NAME         X16    B
```

The length shown for keys in the Additional Third-Party Indexes may vary between versions of Superdex. This is a known situation and does not affect the operation of Suprtool on these paths.

The length of a key is returned by dbinfo-833 and Suprtool will report in the form command the length of a stand-alone key. (SI-PRODUCT-NO is the path that is now supported) Superdex has a variable that impacts the length that is returned for a given key, and that is SICOGNOS. Suprtool will account for the length difference on the chain command but will show the unadjusted length. Normally Superdexs dbinfo-833 would show a length of 12 for a Zoned-decimal tpi-key with a length of 8, if SICOGNOS is set to 1, then Superdex will return 8.

B-tree Support

The Form command has been enhanced to show whether there are B-trees that the Suprtool Chain command can utilize. Form Sets shows a type of "i" for datasets that have B-tree indexes. Currently the Chain command utilizes only those B-trees that are type X or U. The Form command does not show B-trees that it cannot utilize, even if the B-tree does exist.

The Form command also reports a damaged B-tree.

```
>form sets
Database: CD.DB.DATA
          BtreeMode1 = ON       Wildcard character : @

             Set       Item            Entry  Load   Entry
Sets:        Num Type  Count Capacity  Count  Factor Length  B/F
 M-ARTIST    1   M  i  2     211       3      1  %   75      4
 A-SONGS     2   A  i  1     307       41     13 %   20      16
 A-CD-TITLE  3   A  i  1     211       3      1  %   20      16
 D-SONGS     4   D     6     1004      41     4  %   78      4

>form m-artist
Database: CD.DB.DATA
          BtreeMode1 = ON       Wildcard character : @

  M-ARTIST       Master    Indexed      Set 1
     Entry:                    Offset
        ARTIST               X30     1  <<Search Field>>
        NOTES               2X60    31
  Capacity: 211 (4)  Entries: 3  Bytes: 150

>form d-songs
Database: CD.DB.DATA
          BtreeMode1 = ON       Wildcard character : @

  D-SONGS          Detail               Set 4
     Entry:                    Offset
        CD-NO               J2      1
        CD-TITLE            X40     5  (!A-CD-TITLE) <<Indexed>>
        SUB-TITLE           X40    45
        ARTIST              X30    85  (M-ARTIST) <<Indexed>>
        SONG-NO             J1    115
        SONG-TITLE          X40   117  (A-SONGS) <<Indexed>>
  Capacity: 1004 (4)  Entries: 41  Highwater: 41  Bytes: 156
```

Formout File

The Form command writes all output to the file Formout. This file defaults to $stdlist. You can redirect this file to a line printer or a disc drive. If you redirect the Formout file to a disc file, Suprtool assumes a temporary file by default.

```
>:file formout;dev=lp
>form sets                              {writes to line printer}
>:file formout;dev=disc
>form d-inventory                       {writes to temporary disc file}
```

Default Form

If a Chain, Get, or Input command of a self-describing file has been entered, a Form command without parameters shows the fields in the current input source. If a Base command has been specified, but no input source, a Form command without parameters does a Form Sets.

Form Keywords

The Form command shows items, paths, and sets before it searches for a dataset or file with these names. Use a string (e.g., "sets") to display the form of a dataset or file that matches one of the Form keywords.

```
>form "paths"                                    {paths is the name of a dataset}
```

Get Command [G]

Selects an IMAGE dataset from a previously opened Base as the input source for the next extract. You must have read access to all fields in the dataset entry. Only one Chain, Get, or Input command is allowed per extract task. Get always reads the dataset serially.

GET *setname* [(*startrecord*/[*endrecord*]) | (#*count*)]

(Default: all records)

Dataset Input

The first example shows the most common use of the Get command. An input dataset is specified as input to Suprtool. We select a subset of the entire input dataset using the If command:

```
>base store,5,READER                {open for read access only}
>get d-inventory                    {serially read dataset}
>if unit-cost<10000                 {the UNIT-COST field is}
>output out1                        {    automatically defined by Get}
>xeq
```

Selection by Record Number

The (*startrecord*/*endrecord*) parameter permits selection of input records on the basis of the IMAGE record numbers. These numbers always start with 1, and the *endrecord* parameter is assumed to be the last record in the dataset if it isn't specified.

Note that you should use extreme care when you are using the record number selection option on master datasets. This is because the record numbers of master dataset entries can be changed as entries are added or deleted from the dataset. For example, if there are deleted entries in the master dataset, then you could get fewer records than expected.

Numrecs is another way to select only a few records.

When debugging software, it is convenient to scan the first few records of a dataset. Specifying a *startrecord*/*endrecord* parameter makes this easy:

```
>get m-product(1/20)                {first twenty records}
>list                               {produce a formatted list of each …}
>xeq                                {… record with no output file}
```

This example gets any records that are in the first twenty IMAGE locations. This may be fewer than twenty records, if there were deleted or unused entries in the first twenty record numbers.

Random Selection

The #*count* parameter selects every "nth" record from the dataset, where "n" is equal to *count*. This option is designed to allow "random" selection from the dataset. It cannot be combined with the (*startrecord/endrecord*) option.

Test databases can be constructed from random samplings of production databases. Using the #*count* parameter and the Put command we build a test dataset:

```
>base store,5,READER                {open for read access only}
>get d-inventory(#15)               {every 15th record is read}
>put d-inventory,test               {put to the d-inventory dataset in …}
>exit                               {… Test database}
```

Using DBGET

By default, Suprtool always attempts to read the dataset using its own high-speed routines. These are the cases where DBGET is used instead:

1. When the database is open with mode-3.
2. When you specify the Delete or Update command and the input source is a master dataset.
3. When you specify the Delete or Update command and the input source is a detail dataset, but no If command is specified.
4. When Set Privmode Off.

In these cases, Suprtool prints the warning:

```
Using DBGET for the input records
```

How Many Records Does Suprtool Read?

When you Get from a master dataset, Suprtool reads the capacity of the dataset. When you specify a detail dataset, Suprtool reads records until the highwater mark.

If Set Eofread is on then, Suprtool will read to the capacity of the dataset for both master and detail datasets.

To insure that Suprtool reads to the capacity for both Master and Detail datasets, then you should put Set Eofread On in your Suprmgr.Pub.Sys file.

If there is a big difference between the number of entries in the dataset and the highwater mark, Suprtool could take much longer to read the dataset (this is true for all programs, not just Suprtool).

You can use the Form and Verify commands to determine how many records Suprtool will read. The Form command shows the number of entries in the dataset. After specifying a Get command for a detail dataset, use Verify Numrecs to see how many records Suprtool will read:

```
>get d-sales                        {detail dataset}
>form                               {shows #entries}
>verify numrecs                     {#records Suprtool will read}
```

If there is a large difference between the number of entries and the highwater mark, you may wish to repack the detail dataset. If the dataset is dynamic with many entries being added and deleted, it is better to let Suprtool take longer (otherwise you would be continuously repacking the dataset).

For regular monitoring of all your datasets, use HowMessy which shows both the number of entries and the highwater mark for detail datasets.

FLIMIT Different

The Get command may print an error "FLIMIT different than calculated" or a warning "FLIMIT greater than calculated". Suprtool verifies that the physical MPE file that corresponds to the dataset matches the information returned by DBINFO. Suprtool computes the number of records in the MPE file as follows:

#records = (capacity - 1) / blockfactor + 1

where capacity and blockfactor are reported by the Form command. When this error occurs, Suprtool prints the MPE file name, MPE FLIMIT, dataset name, dataset number, and the computed FLIMIT:

```
>get m-test
Error: Dataset with FLIMIT different than calculated!
MPE Filename: TEST01.STORESC.DB   Actual MPE Flimit: 1
IMAGE Dataset: M-TEST (#1)        Calculated Flimit: 2
```

This error is usually caused by restoring an earlier version of the dataset from a backup tape.

Remote Databases

When trying to access a database on a remote computer, the Get command may fail with the error "Privileged File Violation (FSERR 45)". See the discussion under the Base command for workarounds to this problem.

Help Command [H]

Show what commands and options are available in Suprtool.

HELP [*command* | *keyword* [,*option*]]

(Default: browse through the entire help file)

Command Help

If you specify any parameters, Help first assumes that you want help on a Suprtool command. If you know the structure of the help file, you can additionally specify one of the keywords under the command name.

`>help ext`	{help on the Extract command}
`>help ext,notes`	{Notes section of the Extract command}

Keyword Help

If Suprtool cannot find any help in the "Commands" section of the help file, it assumes that you specified one of the outer-level keywords in the help file. To see this list of keywords, type help with no parameters. You see a short introduction to Suprtool and then a list of keywords. You can specify any of these keywords on the Help command. You can also specify a subkeyword.

`>help start`	{Quick Start section}
`>help start,task`	{Task section of Quick Start}

Quick Help - HQ

HQ asks Suprtool to look under the keyword Quick in the help file. Quick contains the text from the Suprtool Quick Reference Guide, offering the experienced user a quick review of the syntax of any command.

`>hq input`	{quick description of Input}
`>hq commands`	{quick list of command names}

Notes

If no parameters are specified, Help allows you to browse through the "help" file. The Help Command uses the QHELP subsystem to allow you to look at the material in the file Suprtool.Help.Robelle (which contains most of the user manual). For "help in help", type "?" when you see the QHELP prompt character ("?"). The help file is organized into levels. To go back to the previous level, press Return instead of a key name. If you press F8, you will exit the QHELP subsystem and return to Suprtool.

If Command [IF]

Specifies a subset of records to select from the input source during the next extract task. The If command supports full logical expressions, with comparisons between all data-types, between data fields and constant values, or between one data field and another. The If command also provides partial string compares, bit field extracts, subscripted IMAGE fields, AND-OR-NOT operators, and parentheses to override precedence. You can use arithmetic expressions involving any numeric data-types.

IF *expression*

Note: The examples below show multiple If commands. These are for illustrative purposes only. Suprtool does not permit multiple If commands in a single task. Instead you can combine multiple conditions using AND and OR.

Alternatives to the If command

There are a few selection criteria that the If command cannot perform. In these cases, you need to use other Suprtool commands. If you wish to select by record numbers, use the record number options in the Get or Input commands. If you wish to limit the number of records selected, use the Numrecs command. See also "Chain Command [C]" on page 137 for an alternative to the Get/If combination of commands.

There Is No Else Clause

The If command in Suprtool does not have an Else clause but the output command does. To select the records that do not match the If criteria, you can use output,else to get both the selected data and the data that was not selected by an if criteria. See the output command for details.

```
>get    dataset                          {this task is the "If … then"}
>if     expression
>output file1,else,link                  {file1 and temp file else created}
>xeq
```

You can also use a second task with the same criteria negated by a NOT.

```
>get    dataset                          {this task is the "If … then"}
>if     expression
>output file1,link
>xeq

>get    dataset                          {this task is the "else"}
>if     not (expression)
>output file2,link
>xeq
```

Expressions

An *expression* specifies the logical criteria that Suprtool uses to select records from the input source.

Simple Expressions

The simplest *expression* is a single *comparison* between two fields (e.g., A=B) or a field and a constant (e.g., A="XX"):

field relation *field*

field relation *constant*

Fields

A field can be a temporary, Defined field, or a field from a self-describing file, or an IMAGE field (with possible subscripts; see below). Each *field* has a type (see "Key Command [K]" on page 220 for further details). The *constant* must match the type of the *field*. If the *field* has a byte-type, you must surround the *constant* with quotes.

```
>if name="TAMMY ROSCOE"                {name is byte}
>if rating>10000                       {rating is integer}
>if balance=arrears                    {compare two fields}
```

Constants

A *constant* is a value that matches the data-type of *field*. Constants are either a string constant in quotes, a numeric constant, or a date constant specified with $date or $today. See the next section about *Constants* for more details.

Constant	Type
"NATHAN ARMSTRONG"	string constant
12345	numeric constant
$date(00/07/09)	date constant July 9, 2000

Relations

A *relation* is one of the size comparison symbols (Suprtool does not use words like "EQUALS" as in QUERY):

Relational operator	Means
=	equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
<>	not equal to

Complex Expressions

Complex *expressions* can be made by combining the AND, OR, and NOT operators, arithmetic operators (+, -, *, / and mod), and parentheses. The order of precedence of operators, from highest to lowest, is

Operator	Precedence
(...)	Highest.
NOT	Take the opposite.
AND	Both must be true.
OR	One or the other must be true.
-	Unary minus.
* /	Higher than addition and subtraction.

+ -	Lowest.

Use parentheses where necessary to change the order of evaluation.

```
>if status="1" and amount>100 or purchased="000115"
>if (status="1" or status="2") and amount>100
```

Multiple Values

You can check a data field for several test values without using the AND and OR operators. After the equals or not-equals sign, list the alternate values separated by commas.

The OR operator is = (equal sign). Instead of "IF A=5 OR A=6 OR A=7", use "IF A=5,6,7". This selects a record if A is equal to 5, 6, or 7.

The AND operator is <>. Instead of "IF A<>5 AND A<>6 AND A<>7", use "IF A<>5,6,7". This selects a record if A is anything but 5, 6, or 7.

```
>if field = 5,6,7
>if part = "12345","67890","39201","92308","14892"
>if delivered <> 981231,990101
```

This method works well if you are searching for a small number of values. Use the $lookup function to check a data field for many test values.

IF $LOOKUP(*tablename*, *fieldname*)

The $lookup function returns TRUE, if the specified field name contains a value from the specified table. You can also look for values that are **not** in a table.

IF NOT $LOOKUP(*tablename*,*fieldname*)

See the Table command for a complete description of how to combine tables and the $lookup function.

$lookup parameter	function
tablename	The name of a table specified in the Table command.
fieldname	A field from the input record. This field cannot be real-type. It must be exactly the same length as the item used in the Table command.

Multiple Values and Table Data

Suprtools can use the “data” loaded into a Table in a comparison operation. The $lookup function will return the data value from the table to compare against another field or literal.

```
In file1sd
Table mytable,char-field,data,tabfile,data(id-field)
If $lookup(mytable,char-field,id-field) = int-field
```

So, what Suprtool will do in this case is read a record, lookup the record in the table and retrieve the data item in the table. If Suprtool does NOT find an entry in the table, a zero will be returned if the data type is numeric and spaces will be returned if it is a byte type.

So, using the case above, if no entry is found in the table, zero is returned, and if int-field is equal to zero, then the record will qualify.

If you don’t want to have any values returned from the table lookup, you just preface the if with a standard lookup.

```
>get    ord-details
>table  cust-table, cust-no, file, custlist,data(state-code)
>if     $lookup(cust-table,cust-no) and &
       $lookup(cust-table, cust-no, state-code) = state-code
>output orders
>xeq
```

When using $lookup to return data, the $lookup must always be on the left side of the expression. If not Suprtool will stop with an error:

```
>if id-field=$lookup(mytable,char-field,id-field)
Error:  $lookup in this context (data comparison) must be on left side
```

Performance of $Lookup

Short-circuit evaluation means that the If command does not always need to evaluate all the parts of the command.

Due to the nature of the $lookup function, it can be, at times CPU intensive, however, since the If command uses short-circuit evaluation, $lookup should be specified as the last part of the If command. For example,

```
>if status = "10" and $lookup(cust-table,account)
```

is faster than

```
>if $lookup(cust-table,account) and status = "10"
```

because Suprtool can evaluate `status = "10"` faster than `$lookup`. When the status is not "10" Suprtool knows the record will not be selected, therefore there is no need to do the $lookup.

$Null(fieldname)

The If $null(*fieldname*) command selects any rows that have null values in them. This feature is available only for SQL databases and only on columns that allow null values:

```
>if $null(SALESTOTAL)
```

If you want to find only those values that are not null, you can add the NOT keyword in front of $null:

```
>if not $null(SALESTOTAL)
```

Constants

This section describes numeric and string constants. See also *Date Selection*.

Numeric Constants

Numeric constants are not enclosed in quotes. Numeric constants may be just simple whole numbers (e.g., 5, 0, -56, 10004) or they may have a decimal point (e.g., 5., 0.0, -.56, 99.9, 1.4). Real and IEEE numbers may also have a scale factor (e.g., 5E-5, 0.01E+4). "Over-punches" for the sign are not required, or recognized, in Suprtool. Always enter -11 as -11, not 1J for a DISPLAY field.

String Constants

String constants are delimited with double- or single-quote marks. That is, either "VANC" or 'VANC'. Any characters within quotes are not upshifted. If the constant is shorter than the field to which it is being compared, the constant is padded with blanks. String constants are expected for fields of type BYTE, U, or X, but numeric constants are expected for fields of type Z (zoned decimal).

```
>if field = " "                          {check for all blanks}
>if field = "XX"                         {double-quotes are okay}
>if field = 'XX'                         {so are single-quotes}
```

If you want to compare for a quote itself, you include two quotes in the string for each quote you want.

```
>if field = "AB""CD"                     {look for AB"CD}
```

Character Constants

Use the ^-character to specify any ASCII character. The number (the actual ASCII value), or letter (^A means control A), must follow immediately after the ^-character. Suprtool treats character constants as strings. When you compare the constant to a field longer than one byte, Suprtool pads the constant with spaces.

```
>define field,1,1                        {byte field}
>if field = ^0                           {binary zero}
>if field = ^G                           {Control-G (bell)}
>if field = ^27                          {escape}
>if field = ^252                         {Roman-8 box}
```

To look for "null values" or "low values" in byte-fields, it is usually sufficient to check the first byte for a binary zero:

```
>define first-byte,bigfield,1,byte
>if first-byte = ^0
```

Subscripts

Use subscripts to access individual items in repeated fields, or to access substrings.

Numeric Subscripts

For repeated numeric fields only one index is allowed.

If Table has the form 10J2, it holds ten double integers.

- Table(1) is the first sub-item.
- Table is the same as Table(1).
- Table(5) is the fifth sub-item.

```
>if table(5) = 23
>if table(2) = 20 or table(4) = 30
>if table(8) = 31 and table(9) = 28
```

Character Subscripts

Character string fields may have 1, 2, or 3 subscripts after them. Character string fields are allowed more than one subscript value.

If ADDR has the form 5X30, it consists of 5 substrings of 30 characters each.

- ADDR(1) is the first 30-character sub-item of ADDR.
- ADDR without subscript is the same as ADDR(1).
- ADDR(2) is the second 30-character sub-item of ADDR.
- ADDR(2,4) is the second sub-item, starting with the 4th byte and extending for the remainder of the sub-item, 27 bytes.
- ADDR(2,4,6) starts at the same location, but extends for only 6 bytes.

If NAME has the form X50, it is not a repeated field.

- NAME is the same as NAME(1).
- NAME(1,4,6) is the first (and only) sub-item, starting at the 4th byte and extending for 6 bytes.
- NAME(1,10) is a field that starts at the 10th byte and implicitly extends to the end of the field (for the remaining 41 bytes).

```
>if name(1,4,6) = "HAWAII"
>if addr(3) = "VANCOUVER, B.C."
>if addr(3,11,20) == "@B.C.@"                {pattern matching}
```

Numeric Expressions

Bit Selections

The If command can extract and test any series of one or more contiguous bits in a field. Suprtool allows bit extracts only on Integer or Logical fields of two bytes in length (one 16-bit word). To do a bit extract from another type of field, first use Define to redefine the data as a two-byte Logical field.

Once Suprtool extracts a bit string, it always treats it as an Unsigned Integer, a Logical, and never interprets it as negative. The format for bit extracts calls for a starting bit number and a bit count. The 16 bits in a computer word are numbered from the left, 0 to 15. The two bytes to extract from need not be on a "word boundary" (i.e., they can start in any byte position). See "Define Command [D]" on page 141 for how to define a two-byte logical field.

field . (*startbit* : *bitcount*)

```
>define bitfield,name,2,logical
>if bitfield.(4:2)=3
```

How to Check a Byte for a Numeric Value

Because Suprtool does not have one-byte integers, it can be difficult to check a single byte for a specific numeric value. Use a two-byte integer Define field and the bit-extract operator to solve this problem:

```
>define word,transcode,2,integer
>if word.(0:8)=13
```

See "Character Constants" on page 195 for an alternate method.

Decimal Places

Use the Item command to specify the number of implied decimal places in an item. If you do not do this, you must scale all numbers in the If command. For example, let's assume that you want to find all inventory records with a cost equal to $80.59. If you do not use the Item command, your If command would look like this:

```
>if cost = 8059                              {no decimal places}
```

By telling Suprtool about the number of decimal places in the cost item, your If command looks more natural (which usually means you will make fewer mistakes):

```
>item cost,decimal,2
>if cost = 80.59                          {decimal places included}
```

Numeric Conversion

The If command can compare two numeric fields to each other (not just one field to a constant). All relation operators are supported: <, <=, =, <>, >, and >=. However, you cannot compare a byte-field to a numeric-type field.

Suprtool usually converts the field on the left side of a relational operator to floating-point. Then the floating-point number is converted into the type of the field on the right side of a relational operator and the comparison is done. The exceptions to this rule are integer-to-double, packed-to-packed, and display-to-display comparisons, which use a direct comparison algorithm.

Truncation errors can occur when Suprtool converts from one field type to floating-point. See also *Accuracy* and *Numeric Truncation*.

Arithmetic Expressions

You can specify arithmetic expressions for any numeric data-type in the If command. Arithmetic expressions involve the operators +, -, *, / and mod. The Mod operator returns the remainder between a dividend and a divisor. Arithmetic expressions cannot start with a numeric constant (e.g., if 2 + a = 10 is invalid). Arithmetic is not allowed on byte-type fields. If you have a byte-type field that consists entirely of numeric digits, redefine the field as display type and use the redefined field name in the If command.

Examples

```
>if field + 10 = 1115                    {numeric field}
>if cost * qty > 10000
>if total < qty * price + tax
>if yymmdd-date / 100 mod 100 <= 03      {first quarter}
```

Division by Zero

Suprtool reports an error and the input record number if an arithmetic computation results in division by zero. Use Set Ignore On to force Suprtool to ignore division by zero errors. With Set Ignore On, the result of division by zero is zero.

Your task executes more slowly if you have a lot of division by zero errors and you have asked Suprtool to ignore them. A better approach is to check for zero in the If expression before using it in division:

```
>if $read
-   qty <> 0  and                        {avoid division by zero!}
-   total / qty > 100
-   //
```

Missing Features

Arithmetic overflow in computations will cause Suprtool to abort.

Accuracy

By default, Suprtool uses floating-point arithmetic to compute. In some cases, there can be slight inaccuracies due to rounding errors. Rounding errors are more likely with Classic floating-point arithmetic (Set Arith Classic). For example, the following expression fails if you have a single integer field with the value 7 or -7:

```
>set arith classic
>if  integer-field / 10 * 10 = 7
```

This expression does not fail when using IEEE floating-point (Set Arith IEEE). If you are developing Suprtool applications for both Classic MPE and MPE/iX, we suggest that you enable Set Arith Classic in Suprtool/iX to ensure exactly the same results in both applications.

Numeric Truncation

The accuracy of arithmetic computations is limited to approximately sixteen digits. Suprtool may truncate four-word integers (quad) or large packed-decimal or display numbers when they are converted to floating-point. Suprtool does not produce any error or warning in this case.

$Abs function

Suprtool supports an $abs function, which returns the absolute value of a number. For example, if a field called Credit contains the value -547.83, the $abs function returns 547.83.

This function will work on a field or even on an expression such as:

```
>if $abs(credit / 100 * 1.07) > 500.00
```

This function will also work in the Extract command:

```
>def newcredit,1,4,double
>ext newcredit = $abs(credit / 100 * 1.07)
```

$Truncate function

Suprtool supports a $truncate function which returns the number to the left of a decimal place. For example, if the field stddev contains the value 547.83, the $truncate function will return 547. Note that there is no rounding.

This function will work fields and expressions:

```
>if $truncate(stddev / 100 * 1.07) > 200
```

This function will also work in the Extract command:

```
>def newdev,1,4,double
>ext newdev = $truncate(stddev / 100 * 1.07)
```

String Expressions

You can do comparisons with byte-type fields in numerous ways using Suprtool. These powerful features minimize the number of tasks you must execute in order to select the data you need. The fewer the number of tasks, the faster your data is delivered to the users and applications that need it.

You can combine byte-type fields together and use the built-in string functions to create string expressions. String expressions involve the + operator and the other string functions such as $lower, $upper, $trim, $ltrim and $rtrim.

Fixed vs. Variable Length Strings

String comparisons are done using fixed- and variable-length strings. For most users, there should be no difference between the two types of strings. When doing string comparisons, Suprtool always pads shorter strings with spaces, with the one exception of comparing two fixed-length fields (see "Byte Fields" below).

String expressions involving the + operator or the $upper, $lower, $trim, $ltrim and $rtrim built-in functions are done using variable-length strings. Suprtool keeps track of the length of every string, and all operations are done using the actual string length. For fields, the length of the string is the length of the field. If you do not want to retain all of the spaces in a field, use one of the built-in trimming functions.

When creating string expressions, string constants are created with the exact length of the constant. For example, the string constant "abc" is three characters long and the string "a" is one.

Byte Fields

For historical reasons, comparing two byte-type fields to each other is a special case. If the two fields are exactly the same length, Suprtool compares them completely. If one field is shorter, the comparison is done for the length of the shortest field. Suprtool does not check for spaces in the trailing characters of the longer field. For example,

```
>define  short, 1,10                    {ten character field}
>define  long ,11,15                    {fifteen character field}
>if      short = long
```

In this example, Suprtool compares the ten bytes in the short field with the first ten bytes of the long field, but ignores the last five bytes of the long field. If the expression on either side of the equal sign consisted of more than one field (using the + operator) or involved any of the string functions, ($upper, $lower, $trim, $ltrim or $rtrim), Suprtool would have compared both sides of the equal sign by padding the shorter field with spaces. It is only the case where you are directly comparing one byte-type field to another that Suprtool uses the length of the shortest field for the comparison.

You cannot compare a byte-field to a numeric-type field. If you have a byte-field that consists entirely of numeric digits, redefine the field as a display-type and use the redefined field name in the If command.

Character Type

Byte-type fields can also be checked to see whether they contain only Alpha, Numeric, Alphanumeric, or Special characters. The complete field is compared against the specified character types.

Type	Characters
Alpha	A-Z, a-z (52 characters)
Numeric	0-9 (10 characters)
Special	anything else (194 characters, including spaces, punctuation, Roman-8 letters, binary junk)
Alphanumeric	A-Z, a-z, 0-9 (62 characters)

For the test result to be true, **all** the characters in the field must be of the specified character type. To test a substring, use the Define command to define a subfield.

```
>if field = alpha
>if field <> numeric
```

Examples:

String	Class
"1234"	numeric
"12.3"	no class, contains both numeric and special
"ABCD"	alpha
"B JONES"	no class, contains both alpha and special
" "	special
"A1B2"	alphanumeric

Pattern Matching

Suprtool can also select records based on a pattern of characters, rather than an exact string of characters. For example, use the following to select all records with "CONNOR" anywhere in the Name field,

```
>if name == "@CONNOR@"
```

The double equals (==) is the operator for pattern matching. The at signs (@) means anything before or after "CONNOR" is acceptable, including nothing.

For character fields, there are two comparison operators for patterns: "==" (matches), and "><" (does not match). The pattern is specified as a quoted string, using the same special characters as in the MPE :Listf command. Embedded spaces are allowed in the pattern and must be matched in the target field.

These are the special characters:

Character	Meaning
@	Zero, or more, characters of any type.
#	A single numeric character.
?	A single alphabetic or numeric character.
~	Zero, or more, blank characters.
&	Escape character to match the next character explicitly (&@ looks for the @ character).
^	Reserved for future use.
!	Reserved for future use.

Any other character must be matched, one for one.

```
>if name=="@ZANDER@"                    {does name contain ZANDER anywhere?}
>if name=='@ZANDER@ARMSTRONG@'          {does name contain ZANDER, perhaps …}
                                        {… other characters, then ARMSTRONG?}
>if name><"@#@"                         {does name not contain numerics?}
>if name=='@qedit@','@suprtool@'        {qedit or suprtool?}
```

For more information, see *Special Characters* in the Glossary.

Finding Special Characters

With the $Clean function you can clean "bad" characters inside of text fields, however the $Clean function does not report back what records were "cleaned". For this reason, we have the $FindClean function. $FindClean will return true if it finds a character defined using the Clean command. This makes it extremely easy to find a set of Special characters that you can define.

```
>in cleansd
>clean "^9","^10"
>if $findclean(nonprint)
>list
```

The above task will list the record if the field nonprint has a Tab (Decimal 9) or a Line Feed (Decimal 10) anywhere in the field. You can Find and clean the "bad" characters from a field at the same time:

```
>in cleansd
>clean "^9:^10"
>if $findclean(nonprint)
>extract nonprint=$clean(nonprint)
>list
```

Trimming Spaces ($Trim, $Ltrim, $Rtrim)

Use one of three built-in string functions to remove leading or trailing spaces from a string expression. The three functions are:

$Trim:	Remove leading and trailing spaces from the string expression.
$LTrim:	Remove leading spaces.
$RTrim:	Remove trailing spaces.

Because Suprtool pads shorter strings with spaces when doing comparisons, trimming spaces is most useful when creating a combined string with several fields. For example, you might want to combine a person's first and last name (including a space between the two):

```
>if $trim(first) + " " + $trim(last) = "Joe Smith"
```

Mixed Case ($Upper and $Lower)

By default, Suprtool does an exact match when comparing two string expressions. If the expressions vary in the capitalization of characters, Suprtool finds them to be different. To do caseless string comparisons or pattern matches, use the $upper or $lower functions. Both ASCII and Roman-8 characters are shifted by $upper and $lower. For example,

```
>if $upper(city) = "VANCOUVER"
>if $lower(city) = "edmonton"
```

Note that if you use the $upper or $lower functions, Suprtool does not upshift or downshift any constants used in the comparison. You must explicitly specify the constants in the correct case, or you can use $upper or $lower with the constant:

```
>if $upper(city) = $upper("vancouver")
```

Use the $upper or $lower functions for caseless pattern matching. As with other comparison operators, you must specify constants in the correct case when doing pattern matching:

```
>if $upper(city) == "VAN@"
>if $lower(city) == "ed@"
```

You can use $upper and $lower with string expressions that combine many fields and string functions as shown in the following example:

```
>if $read
-   $upper($trim(first) +
-              " "              +
-              $trim(last))
-   = "JOE SMITH"
-   //
```

Date Selection

The If command has four functions to help select records based on dates: $date, $today, $days and $stddate. The $date function works for any date. The $today function works for the current date and dates relative to today. The $stddate and $days functions work for almost any date. To use these date functions, you must first identify the date format of an item by using the Item command.

The $date function makes it easier to specify a target date for certain date formats (e.g., PHdate or ASK). To select records based on a specific date, use this feature:

```
>if field=$date(year/month/day)
```

Suprtool checks the date's validity. To select the transactions for January 1999, you would do the following:

```
>item trans-date,date,phdate
>if trans-date >= $date(1999/01/01) and &
    trans-date <= $date(1999/01/31)
```

Relative Dates

You can specify a relative date using the $date function. Then you can create job streams that don't rely on hard-coded dates. The general syntax of the $date function is:

$date(*year*/*month*/*day*)

The *year* can be a specific number (e.g., 2000) or an asterisk "*" for the current year. To specify a relative year, you add or subtract years from the one you specified:

```
>if field=$date(2000/01/01)          {January 1, 2000}
>if field=$date(2000-1/01/01)        {January 1, 1999}
>if field=$date(*-1/01/01)           {January 1, last year}
```

The *month* can be a specific number (e.g., 6 for June) or an asterisk "*" for the current month. To specify a relative month, you add or subtract months from the one you specified:

```
>if field=$date(2000/06-1/01)        {May 1, 2000}
>if field=$date(*/*/01)              {start of current year and month}
>if field=$date(*/*-1/01)            {start of last month}
>if field=$date(*/*-18/*)            {exactly eighteen months ago}
```

The *day* can be a specific number (e.g., 15), an asterisk "*" for the current day, the word "first" for the first day of the month, or the word "last" for the last day of the month. You cannot add or subtract relative days; use $today instead.

```
>if field=$date(2001/01/first)       {January 1, 2001}
>if field=$date(*/*/*)               {today's date}
>if field=$date(*/*-1/last)          {last day of previous month}
```

Combining these features makes it possible to generate batch jobs that require no operator input. For example, to select all of the transactions for last month you would use:

```
>item trans-date,date,phdate
>if trans-date >= $date(*/*-1/first) and &
    trans-date <= $date(*/*-1/last)
```

Month End

Suprtool is always expecting a valid date. Suppose that you have a month-end job that contains the following If command:

```
>if field = $date(*/*-1/*)
```

When you run the job on May 31, 2000, if Suprtool were to use the literal interpretation of $date(*/*-1/*), it would use the date April 31, 2000. In fact, there is no such date; April has only 30 days. Whenever you specify * for the day, and the day is greater than the last day of the month you specified, Suprtool uses the actual last day of the month instead of the current day of the month. In our example, Suprtool would use April 30, 2000. Suprtool will take leap years into account when calculating the last day of February.

Today's Date

To select records based on today's date, use the following:

```
>if field=$today                    {today's date}
>if field=$today(-1)                {yesterday's date}
>if field=$today(+1)                {tomorrow's date}
```

Use the Item command to qualify the field as a date. The $today function accepts one optional argument which is the number of days before or after today. The maximum number of days in either direction is 9999.

yymmdd and Beyond 1999

Because dates spanning the turn of the century will not collate properly for the yymmdd form, you need to use $stddate to compare these dates.

```
>item ship-date,date,yymmdd
>if ship-date < $date(2000/12/31)             {will not work}
>if $stddate(ship-date) < $date(2000/12/31)   {will work}
```

Finding Invalid Dates

Use the $invalid function to find invalid dates. An invalid date is a number in a date format whose date equivalent cannot be found on a calendar. For example, a month value of 99 would be considered invalid.

```
>base store.demo
Database password [;]?
>get d-sales
>item deliv-date,date,ccyymmdd
>if $invalid(deliv-date)
>out baddates,link
>xeq
```

$Stddate Function

The $stddate function converts any date format in nearly any data-type container and internally converts it to the ccyymmdd format in a double integer container.

This allows you to compare dates with dissimilar formats and data-types. For example,

```
>in orddets
>item order-date,date,ccyymmdd
>item bill-date,date,mmddyyyy
>if $stddate(bill-date) <= order-date
>output badords,link
>xeq
```

This feature is also available for dates that have two-digit years. The century portion of the date will be generated by $stddate, which uses the normal cutoff rules specified by Set Date Cutoff.

```
>in invdets
>set date cutoff 20
>item invoice-date,date,yymmdd
>item close-date,date,mmddyyyy
>if $stddate(close-date) <= $stddate(invoice-date)
>out badinvs,link
>xeq
```

In this case all invoice-date values with a *yy* portion between 20 and 99 will have a 19 for the century. All invoice date values with a *yy* portion of less than 20 will have 20 for the date generated by the $stddate function.

Invalid Dates

A date must be valid before $stddate can convert it to the ccyymmdd format. Otherwise, a value of 0 will be returned for any invalid dates. An invalid date is a number in a date format whose date equivalent cannot be found on a calendar. This includes dates selected by the $invalid function. We can eliminate the invalid dates from the above task by changing the If command slightly.

```
>get invoice-detail
>set date cutoff 20
>item invoice-date,date,yymmdd
>item close-date,date,mmddyyyy
>if (not $invalid(close-date) &
 or  not $invalid(invoice-date)) &
 and     $stddate(close-date) <= $stddate(invoice-date)
>out badinvs,link
>xeq
```

In this example, if either the close-date or the invoice-date are invalid, then they will not be evaluated by the $stddate function and will not be selected. Although your requirements may be different, you need to remember that invalid dates evaluated by the $stddate function will return a 0 value.

$Days Function

Suprtool supports a $days function, which converts any supported date to a Julian Day number (the number of days since 4713 BC). This allows for Date arithmetic, in which you can calculate the difference between two dates, even if they have dissimilar formats.

For example, you could find all orders that were not shipped within 30 days of being ordered.

```
>form ordfile
    File: ORDFILE.SALES.MFG       (SD Version B.00.00)
       Entry:                  Offset
          ORDER-DATE        X8      1        <<CCYYMMDD>>
          SHIP-DATE         X8      9        <<MMDDYYYY>>
          ORDER-NUMBER      X6      17
    Limit: 10000  EOF: 15  Entry Length: 22  Blocking: 16

>in ordfile
>if $days(SHIP-DATE) - $days(ORDER-DATE) >=30
>list
>xeq
```

Invalid Dates

As with the $stddate function, if a date is not a valid date, then the result of the $days function will be zero. In the example above, if the order has not yet been shipped, then the SHIP-DATE will likely be blank, or zero, or some other special value. $Days(SHIP-DATE) will be zero, and the resulting calculation will be a negative number.

Notes on Relative Dates

The $date and $today functions always generate a constant from the date, just as if you had typed it. For example, when run on February 13, 2001, the following task:

```
>item field,date,yymmdd
>if field > $today
```

is the same as:

```
>if field > 010213
```

Suprtool normally does no date conversion of the actual dates. Dates that do not start with the year do not collate correctly, so Suprtool does not allow relative comparisons with them (<, <=, >, and >=), although you may still compare for strict equality or inequality. The following examples will be rejected by Suprtool:

```
>item trans-date,date,ddmmyy
>if trans-date >= $date(*/*-1/first) and &
    trans-date <= $date(*/*-1/last)
   Error: Invalid date format for the comparison

>input myfile
>define mydate,1,6
>item mydate,date,ddmmyy                {e.g., 301100}
>define ....
>if mydate > $date(00/11/01)
   Error: Invalid date format for the comparison
>if mydate > $date(01/11/00)
   Error:  Invalid date:  Year = 1 Month = 11 Day = 00
```

Although you cannot do relative date comparisons against non-collating dates in their normal format, you can convert the date into ccyymmdd format using the $stddate function. This date format will collate and, therefore, you can do relative date comparisons using the $stddate function.

```
>item trans-date,date,ddmmyyyy
>if $stddate(trans-date) >= $date(*/*-1/first) and &
    $stddate(trans-date) <= $date(*/*-1/last)
```

If the date format does not allow the specification of a certain day, such as yymm, ccyymm, yyyymm, aamm, ccyy and mmyyyy, then you do not need to specify the entire date format, although Suprtool will allow either format for $date.

```
>item trans-month,date,yymm
>item purch-date,date,yymm
>if trans-month <= $date(*/*/*) and &
    purch-date >= $date(00/01)
```

Because dates beyond 1999 in the yymmdd and yymm date types do not collate correctly, relative comparisons are no longer valid. Suprtool produces an error in the following case:

```
>item trans-date,date,yymmdd
>if trans-date >= $date(2001/01/01)
Error: Cannot use a date beyond 1999 for this date format.
```

You can override this setting by entering the Set Date Ifyy2000error command:

```
>set date ifyy2000error Off
>item trans-date,date,yymmdd
>if trans-date >= $date(2001/01/01)
```

Century and $Date

Suprtool needs to generate a $date or $today date in the ccyymmdd format. If you specify a two-digit year in the $date function, Suprtool needs to assume a century for the given date:

```
>item trans-date,date,ccyymmdd
>if trans-date >= $date(01/01/01)
```

Suprtool assumes 20 for the century if the specified year is less than the Set Date Cutoff value, and 19 if the specified two-digit year is greater than or equal to this value.

Oracle Dates

Oracle dates contain both the date and the time. The $date and $today functions check the date, but ignore the time.

Date Limits

The $date function in Suprtool can generate dates between the years 1583 and 2583. Some date formats have limits based on their particular format, such as 2027 for a Calendar date and 2259 for the aammdd aamm, mmddaa, ddmmaa dates.

Non-Collating Date Types

You can use the $stddate function to convert the non-collating date format to a J2 data item with a date format of ccyymmdd.

For example, to select the purchases by the field purch-date for November 2000 in a ddmmyy X6 field, you would use the $stddate function as follows:

```
>item purch-date,date,ddmmyy
>if $stddate(purch-date) >= $date(2000/11/first) and &
    $stddate(purch-date) <= $date(2000/11/last)
```

Dynamic Date Selection

Using a first pass to generate a Suprtool command dynamically, then using that command in a second pass, is a powerful technique.

You can use the If command for dynamic date selection. Suppose you have a control file that maintains the start and end of a range of dates in which you are interested. You can use the control file to select records from another file or dataset, based on this date range. This is a two-step process, in which the first Suprtool pass creates the If command with your dates, and the second pass does the actual selection from the dataset.

```
>input datecntl                              {read the one-line control file}
>define start-date,1,6,byte                 {start date is the first six bytes}
>define end-date,7,6,byte                   {end date is the next six bytes}
>extract "if sales-date >= '"               {assemble the If command}
>extract start-date
>extract "' and sales-date <= '"
>extract end-date
>extract "'"
>output seldate,temp                        {write the If command to a file}
>xeq
```

This task produces a usefile that looks like this:

```
if sales-date >= '001101' and sales-date <= '001231'
```

Now you can use this file to do the actual selection:

```
>base salesdb,5,password
>get sales-detail
>use seldate                                {use the file that has the If command}
>output sdetail
>exit
```

Adjust Batch Jobs for Holidays

In automating production batch jobs, it would be very nice to know, at the MPE command level, whether today is a holiday (or yesterday or tomorrow). You can use

```
if hpday = 1 then
```

to check for Sunday, but you can't use

```
if hpday = holiday then
```

Everyone has their own schedule of holidays. Here is how we cope with this problem in our backup job, using Suprtool to compare yesterday's date to a list of holidays for the year:

```
!
!comment  Occasionally the backup is not run.  This is
!comment  usually because of holidays, when nobody is in the
!comment  office.  The file SKIPDAYS.STREAMS.PROD contains a
!comment  list of holidays in YYMMDD ASCII format.  Because
!comment  the backup job runs after midnight, it checks if
!comment  YESTERDAY's date is in the SKIPDAYS file.  If it
!comment  is, the SKIPTODAY JCW is set to 1.
!
!run suprtool.pub.robelle
 input  skipdays.streams.prod
 define skipdate,1,6
 item   skipdate,date,yymmdd
 if     skipdate = $today(-1)
 output $null
 exit
!setjcw skiptoday = suprtooloutcount
!
!if skiptoday > 0 then
!  tellop --- Yesterday was a holiday.  No Backup today.
!else
!  comment  Insert the BACKUP commands here:
!  comment      STORE, etc.
!endif
```

The skipdays file is a regular text file with records in the following format:

```
----+----1----+----2----+----3----+----4
000101    New Years Day
000414    Good Friday
000524    Victoria Day
001009    Thanksgiving (Canada)
001111    Remembrance Day
001225    Christmas
001226    Boxing Day
010101    New Years Day
010405    Good Friday
010520    Victoria Day
011014    Thanksgiving (Canada)
011111    Remembrance Day
011225    Christmas
011226    Boxing Day
```

Long Expressions

Long If commands can use an ampersand to continue the command over several lines:

```
>if status = "20" and   &
>>  state = "AZ",       &
>>          "CA",       &
>>          "OR"                        {no ampersand on the last line}
```

This is awkward to use, and, for internal reasons, the maximum length is restricted to 256 characters. The $read function makes it easier to enter long If commands. Its maximum length is based on the complexity of the expression, not on the number of characters.

$Read Function

The $read function reads the If expression from $stdinx, or from the usefile if the If command is in a usefile. $Read continues to prompt for input lines until you press Return or enter "//." You must remember to enter all the necessary parts of the If expression, including connectors like AND and OR and commas. You do not use an ampersand (&) to continue from one line to the next when using $read.

```
>if $read                           {prompt for the expression}
-status = "20" and                  {$read prompts with "-"}
-state = "AZ",                      {the comma is still needed}
-        "CA",
-        "OR"                       {no comma on the last line}
-                                   {blank line to terminate $read}
```

Redoing $Read

When prompting for an expression, $read saves each line in the redo stack and accepts the Before, Do, Listredo, and Redo commands. This provides an easy way to specify all or part of a previous $read expression.

Error: Data Overflow

While the $read function permits long expressions, there are other internal limits within the If command. The first is a limit on the amount of space for constants. Suprtool must blank-fill all string constants to their full length. The following example overflows the data space:

```
>define char-256,1,256
>if char-256 = "a","b","c"
Error:  Data Overflow
```

In this example, Suprtool attempted to create three 256-byte constants. There wasn't enough room for the last constant. Solutions to this problem include:

1. If possible, define short fields. If you have long field names, you may want to use the Define command to define shorter subfields.
2. Use tables and $lookup for many values.
3. Split the extract task into multiple passes. On the first pass, use an If expression that results in the fewest possible number of output records. Use the output file from the first pass as input to the second. Apply the remainder of your If expression during the second pass.

Error: Code Overflow

Suprtool translates If commands to an internal machine representation. There is a limit on the size of this code. When an error occurs, there is little you can do except use tables and $lookup wherever possible, and when this fails use multiple passes.

$Read in Usefiles

When you specify $read in a usefile, Suprtool expects the If expression to appear in the usefile. This provides a method for storing and executing complicated If commands.

You can also manipulate Suprtool into prompting for portions of an If command. When the If command with $read is the last command in a usefile, Suprtool satisfies the $read from $stdinx. The $read function can appear anywhere in which a space can appear, so you can use this to prompt the user for values.

```
>use prompt.use
>in  sdfile                              {first line of usefile}
>if status=$read and &                   {continue the If command}
    state = $read                        {last line of the usefile}
-"10"                                    {prompt for status}
-//                                      {end of prompt for status}
-"AZ",                                   {prompt for state}
-"CA",                                   {user must remember comma}
-"OR"                                    {user also enters quotes}
-//                                      {end of the second read}
```

Notes

Suprtool is not designed to be used by end users. We prefer that you write intelligent front-ends that understand user applications and hide the details of Suprtool from end users. We recommend that you use $read from usefiles only for one-time tasks, or for tasks used by experienced Suprtool users who do not require a friendly user interface.

$-functions

Some $-functions were originally designed for if command and some are more suited for the extract command. Because of this some $-functions don’t work in the if command, such as $edit, and others such as $number have interesting side effects.

Specifically, with $number, when you work on byte type fields, they will be updated internally to be treated as display:

```
Get dataset
Def display-field,byte-field,display
If $number(display-field) > 10000
Out somefile,link
xeq
```

The field byte-field will have leading zeroes and treated as a display.

Input Command [I]

Opens the file that will be the input source for the next extract task. This file can be an MPE disc file, an MPE tape file, a KSAM file, * for $stdinx, a card file, or some other MPE file with fixed-length records. Suprtool executes the Input command immediately -- it does not wait for an Xeq command before opening the Input file.

INPUT *file* [=*setname*] [(*startrecord*/[*endrecord*])]
[(#*count*)]

(Default: all input records)

Input File

The first example shows the most common use of the Input command. An input file is specified as the input source to Suprtool. We select a subset of the input data with the If command. Before using the If command, we must define a field within the input record:

```
>input dinvent                    {input is from a disc file}
>define a,11,2,int                {"A" is an integer that starts}
>output outfile                   {  at the 11th byte of Dinvent}
>if a<10000                       {records with field "A" less than}
>xeq                              {  10000 are written to Outfile}
```

Dataset Files

The =*setname* parameter notifies Suprtool that the records have the same internal data structure as the entries in this IMAGE dataset. You can then specify sort/select fields by field name instead of by the physical offset within the record. You must have done a previous Base command, and *setname* must exist in the database. You must have read access to all fields in the dataset entry, and the record length of the file must match that of the dataset entry.

Using the Define command is error-prone, because it forces you to calculate offsets and lengths of each field in your input record. If you used Suprtool to create a disc file from an IMAGE dataset, you do not need to use the Define command:

```
>base store,5,READER              {open for read access only}
>get d-inventory                  {input from a dataset}
>output dinvent                   {automatically defined by Get}
>xeq                              {process this task and continue}
>input dinvent=d-inventory        {same format as d-inventory}
>if unit-cost<10000               {use the actual IMAGE field name}
>sort cust-account                {again, the IMAGE field name is used}
>list                             {list the selected records with}
>xeq                              {  their field names and values}
```

Input from $Stdinx

If *file* is "*", $stdinx is opened as the source of input records. To terminate Input from $stdinx, hit Control-Y on your terminal after the last input record or make ":eod" the last input record in your batch job. Do not use ":eof:" to end the input data as this causes the batch job to abort.

The record size of the output file is determined by the record size of $stdinx. Because this record size varies in terminals and batch jobs, we recommend that you build the output file first and use the Output xxx,Erase option.

```
!job mgr.robelle
!purge outfile                          {create the output file}
!build outfile;rec=-80,16,f,ascii
!run suprtool.pub.robelle
>input *                                {input from $stdinx}
>output outfile,erase                   {output to an existing file}
>exit                                   {end of Suprtool commands}
Line 1 of 2.                            {input data}
Line 2 of 2.
:eod                                    {end of input data}
!listf outfile,2
!eoj                                    {end of the job}
```

Permanent and Temporary Files

If you have both a temporary and a permanent file with the same name, Suprtool tries to open the temporary file.

Selection by Record Number

The (*startrecord*/*endrecord*) parameter specifies a range of input records by record number. The default value for endrecord is the highest record in the file. The first record number is 0 for MPE files and some KSAM files, or 1 for other KSAM files.

The (*startrecord*/*endrecord*) parameter must come after the =*setname* parameter, if it is present. You can check your record selections with the Verify command.

When debugging software, it is convenient to scan the first few records of a file. The *startrecord*/*endrecord* parameter makes it easy to scan these records:

```
>input dinvent(0/19)                    {first twenty records}
>list                                   {produce a list of each record}
>xeq                                    {   in "OCTAL,CHAR" format}
```

$first/$last

The input command has $first and $last mnemonics, which can be used on a range selection of records on the input command. It was designed to handle a request to list the last N number of records in a file as in:

```
Input somefile($last-10/$last)
```

Suprtool will parse the Range selection and semantically check if the record range entered is logical. For instance, $first-2 and $last+10, which do not make logical sense would throw and error. Similarly, if a record only has 5 records in it then $last-10 or $first+7, would also throw an error.

Random Selection

The (*#count*) parameter specifies that every "nth" record in the input file should be selected. This option is designed for random sampling of the input file. The (*startrecord*/*endrecord*) parameter cannot be used with this parameter. Like (*startrecord*/*endrecord*), (*#count)* must come after the =*setname*, if present.

Test files can be constructed from random samplings of production files. We can build a test file with the #count parameter:

```
>input dinvent(#15)                {every 15th record is read}
>output dtest                      {create an output file with every}
>xeq                               {   15th record from Dinvent}
```

Magnetic Tape Input

For Suprtool to read a file on magnetic tape, you must do two things:

1. Issue a :File command for the tape.
2. If more than 10,000 records will be read from the tape, you must specify the Numrecs command.

This example creates a disc file from the records on magnetic tape:

```
>:file tin;dev=tape                {:File command for the tape drive}
>input *tin                        {you must specify "*" on the Input}
>numrecs 30000                     {specify 30000 input tape records}
>output tout                       {we build the output file with}
>xeq                               {   room for 30000 records}
```

The Input command is processed immediately. Suprtool does not wait for an Xeq or Exit command before opening the Input file. If the session seems to hang after the Input command, check the console to see whether it is waiting for the =reply.

Notes

Only one Input, Chain, or Get command is allowed per extract task. The Input command opens the specified file *immediately*. The file is held open until the input is reset or the current task completes.

If at least one sort key has been specified, the Input command is optional, but you can specify the input file by means of a :File command (:File Input=xxx).

Suprtool first attempts to open the input file as READ, SHARED, NO LOCKING. If it cannot open the file because someone else has it open with LOCKING, Suprtool tries again with LOCKING specified. If the input file is a KSAM/V file, Suprtool eventually closes the KSAM/V file and reopens it with LOCKING and INOUT access.

If you specify a self-describing input file with Oracle dates, Suprtool will automatically recognize the Oracle date attribute.

Item Command [IT]

Use the Item command to specify the number of implied decimal points or the date format for an item. The $date and $today functions of the If command work only with dates. The Item command must precede any If or Extract commands.

ITEM *itemname*,DATE | DECIMAL,*attribute*

Itemname

The *itemname* must be an IMAGE itemname, an SD field, or a Defined field. If the item name exists both as an IMAGE item name and as a Defined field, the *attribute* is associated with the IMAGE item name and the Defined field is deleted. If the item name exists both as an IMAGE item name and as an SD field, the *attribute* is only associated with the IMAGE item name and a warning is produced. The item name cannot be qualified with a subscript.

Date Formats

For dates, the *attribute* must be one of the following:

Attribute	Attribute
ASK	ccyymm
Calendar	ccyy
ddmmyy	aammdd
ddmmyyyy	aamm
mmddyy	mmddaa
mmddyyyy	ddmmaa
Oracle	SRNChronos
PHdate	mmyyyy
yymm	yyddd
yymmdd	ccyyddd
yyyymm	HPCalendar
yyymmdd	JulianDay
yyyymmdd	EDSDate
ccyymmdd	PHDate8

Abbreviations

When specifying the Date keyword, you can use a leading subset for the date attribute. For example, if you want to specify the Calendar date type, you can specify only CA.

```
>item cal-date,date,ca
```

If you do not like this feature, you can turn it off by specifying the following command in your Suprmgr file:

```
>set itemabbreviatedate off
```

Data-Types for Dates

Each date *attribute* is compatible with certain data-types. For more information, see the table on data-types in the Define command. The following table shows the compatibilities:

Date-Attribute	Data-Type Compatibility
ASK	J1 and K1
Calendar	J1 and K1
ddmmyy	X6, Z6, J2, K2, and P8 or greater
ddmmyyyy	X8, Z8, J2, K2, and P10 or greater
mmddyy	X6, Z6, J2, K2, and P8 or greater
mmddyyyy	X8, Z8, J2, K2, and P10 or greater
Oracle	X7
PHdate	J1, K1, J2, and K2
yymm	X4, Z4, J1, and K1
yymmdd	X6, Z6, J2, K2, and P8 or greater
yyymmdd	J2, P8
yyyymmdd	X8, Z8, J2, K2, and P10 or greater
ccyymmdd	X8, Z8, J2, K2, and P10 or greater
ccyymm	X6, Z6, J2, K2, and P8 or greater
yyyymm	X6, Z6, J2, K2, and P8 or greater
aammdd	X6
aamm	X4
mmddaa	X6
ddmmaa	X6
ccyy	X4, Z4, J1, and K1
SRNChronos	X6
mmyyyy	X6, Z6, J2, K2, and P8 or greater
yyddd	X5, Z5, J2, K2, and P8 or greater
ccyyddd	X7, Z7, J2, K2, and P10 or greater
HPCalendar	J2, K2
EDSDate	J2, P8
JulianDay	J2
PHdate8	J1, K1, J2, and K2

Date Limits

The $date function in Suprtool can generate dates between the years 1583 and 2583. Some date formats have limits based on their particular format, such as 2027 for a Calendar date and 2259 for the aammdd aamm, mmddaa, ddmmaa dates.

Calendar

The Calendar *attribute* is provided for users who have fields containing the 16-bit MPE Calendar date format as an unsigned, **logical** value with seven bits for the year of the century (bits 0-6), followed by nine bits for the day of the year (bits 7-15). The Calendar date format only supports dates up to the end of the year 2027.

PHdate and PHdate8

The PHdate and PHdate8 *attributes* are compatible with the COGNOS PowerHouse date format. If the *data-type* is J1 or K1, the date is stored as a LOGICAL value with seven bits for the year of the century (bits 0-6), four bits for the month (bits 7-10), and five bits for the day (bits 11-15). If the *data-type* is J2 or K2, the date is stored as yyyymmdd.

PHDate and PHDate8 date formats are similar, however PHDate values for the year range from 0 - 99, whereas PHDate8 year values are from 0 - 127. A year of 0 in PHDate could mean either 1900 or 2000 depending on user applications. A year of 0 in PHDate8 means 1900, and 100 means 2000. The PHDate8 date format is found in PowerHouse version 8.19 and higher.

ASK

The ASK *attribute* is compatible with the ASK manufacturing software. ASK uses a special date format stored as a single integer or a single logical (i.e., J1 or K1 in IMAGE). This date is relative to January 1, 1973.

yyymmdd

The yyymmdd *attribute* is similar to yymmdd, except that the first digit denotes the century. If the first digit is a 1 (one) then the century is 19, and if the first digit is a 2 (two) then the century is 20. Only data-types of P8 and J2 are supported for this date *attribute.*

This date format is used by some third-party software packages such as MACS and APS.

EDSDATE

The EDSDATE date format is similar to the yyymmdd format, in which the first digit represents the century. The first digit in the EDSDATE is either 0 or 1: a 0 represents a century of 19 and a 1 represents a century of 20.

JulianDay

The JulianDay number is the absolute count of the days that have elapsed since January 1, 4713 BC on the Julian calendar.

Typically, "Julian Day numbers" refer to an integer number corresponding to whole days, while the "Julian Date" may mean an integer plus a decimal value that resolves the Julian count to precise parts of a day. Suprtool supports the "JulianDay" number and does not attempt to support an hour or point in the day.

aammdd and Related Date Formats

The aammdd *attribute* is similar to yymmdd, except the aa portion of the date uses a combination of letters and numbers in order to represent dates beyond 1999.

The aammdd date format was developed by James Overman of HP for use in their MM3000 product. This format is available only for X6 data-type.

By substituting a letter of the alphabet in the first position of the year, we can extend a six-digit date and also ensure that the dates collate correctly. For example:

YY of AAMMDD	CCYY
00 - 09	1900 - 1909
90 - 99	1990 - 1999
A0 - A9	2000 - 2009
B0 - B9	2010 - 2019
C0 - C9	2020 - 2029

Because letters are greater than numbers in the collating sequence you can ensure that aammdd dates beyond 1999 will order correctly.

Suprtool also supports other date formats with this two-character year representation. These formats are aamm, mmddaa and ddmmaa.

Oracle

Oracle dates include both the date and the time. The $date and $today functions only apply to the date part of Oracle dates.

SRN Chronos

The Srnchronos date format is used in applications from Software Research Northwest. Like Oracle dates, this format includes the date and time, but the time portion is ignored by the $date and $today functions.

DDD Dates

Dates consisting of ddd in the format name use the ddd to represent the *n*th day in the current year. This means that January 1 will be day 001, and Dec 31 will be day 365 on non-leap years. Some people refer to these type of dates as Julian dates.

HPCalendar

The HPCalendar date format is supported by HP's new HPCalendar intrinsic and consists of a 32-bit integer number, whereby the first 23 bits represent the year and the last nine bits represent the day of the year.

Decimal Places

The decimal *attribute* is the number of implied decimal places in an item. The minimum number of implied decimal places is 0. The maximum is based on the data-type of the item:

Data-Type	Maximum Implied Decimal Places
I1	5
I2	10
I3	15
I4	19
K1	5
K2	10

P*n*	*n-1*
Z*n*	*n*

You cannot specify implied decimal places for byte-, char-, real-, long-, or IEEE-type items.

Once you define a decimal place, almost every command in Suprtool is affected. Suprtool accepts numeric values with decimal points or scales integers according to the number of implied decimal places (e.g., specify two implied decimal places, then enter 1,000 to represent 1,000.00). All formatting commands format fields with a decimal point when appropriate.

Constant Values

When specifying numeric constants for a field with implied decimal places, there are different formats that you can use. For example, assume that we use the Item command to specify two implied decimal places for an amount field. The following are examples of constant values for this item:

Constant	Interpretation
0	zero value padded as necessary
1	$1.00
0.01	$0.01
.01	also acceptable for $0.01

Notes

Compound Items

When you specify a compound item, the attribute applies to all elements of the compound item.

```
>item monthly-totals,decimal,2          {12 occurrences}
>if monthly-totals(5) > 1000.00
```

You cannot apply an attribute to only one sub-item of a compound item:

```
>item monthly-totals(5),decimal,2
Error:  Missing attribute for the Item Command
```

When to Specify the Item Command

The Item command affects almost all other Suprtool commands. It should be used as follows:

- For databases, specify it immediately after the Base command, but before any Extract or If commands.
- For MPE and KSAM files, it is best to specify the Define and Item commands immediately after the Input command.

Usefiles

You can use a usefile as a mini data dictionary for Suprtool. For databases, it is a good idea to put all the Item commands associated with a database into a usefile. After the Base command has been specified, the usefile can describe the items in the database to Suprtool. For MPE files, you can put both the Define and Item commands in a usefile and execute them right after the Input command for the file. If you use the Link output option, both date formats and implied decimal places are saved in the self-describing file, so they never need to be specified again.

Key Command [K]

Specifies the next sort field for an extract task, using an explicit byte position in the record, not a field name. See "Sort Command [SO]" on page 272 for specifying sort fields by IMAGE field name or Defined field, or by field name in an SD file. Up to 20 Sort and Key commands may be specified per extract task; the first is the major sort field.

[KEY] *byteposition*,*bytelen* [,*type*] [,DESC]

(Default: *type*=BYTE, ASCENDING order).

Parameters

Desc specifies that the field is to be sorted in Descending order.

The *byteposition* is a number between 1 and the length of the input records, specifying where the sort field begins. The *bytelen* parameter is a number from 1 to the length of the record, specifying how long the sort field is.

The *type* is a word that gives the desired data-type of the field for sorting purposes:

Type	Description
BYTE	(unsigned, straight compare)
INT/INTEGER	(two's complement)
DOUBLE	(two's complement)
REAL	(Classic floating-point)
LONG	(Classic floating-point)
IEEE	(IEEE floating-point)
PACKED	(packed-decimal)
PACKED*	(packed-decimal, last nibble unused)
DISPLAY	(zoned-decimal numeric field)
LOGICAL	(unsigned, like BYTE, 2 characters)
CHARACTER	(for Native Language Support)

For compatibility with Sort.Pub.Sys, the Key command also accepts FPOINT as the data-type for IEEE numbers.

Examples

The first example sorts an integer (PIC S9(4) COMP) field which starts on the 11th byte of the input record. We sort the entire input file based on one key:

```
>input bigfile                    {input from a disc file}
>key 11,2,int                     {key is an integer that starts}
>output outfile                   {   at the 11th byte of Bigfile}
>xeq                              {create Outfile and prompt for}
                                  {   more Suprtool commands}

>input discfile                   {another input file}
>key 14,4,double,desc             {double integer (PIC S9(9) COMP)}
>output ofile2                    {sort input in descending order}
>exit
```

The following examples show the various data-types and combinations that are available with the Key command:

`>key 1,10`	{byte data-type}
`>key 1,10,desc`	{descending sort sequence}
`>key 11,4,double`	{I2 or J2, S9(9) COMP}
`>key 1,6;11,4,ieee`	{X6 string and an E2 field at byte 11}
`>key 21,6,packed`	{P12, S9(11) COMP-3}
`>key 21,6,packed*`	{P12, S9(10) COMP-3, wasted byte}

Notes

The command name, Key, is optional. Any command that starts with a numeric character is assumed to be a Key command. The Verify command shows all of the current key fields, and the Reset command cancels them. If no sort fields are defined prior to the Xeq or Exit command, Suprtool performs a copy, not a sort.

Link Command [LIN]

Send commands to Suprlink.

LINK [*suprlink-command*]

Switching into Suprlink

If you enter Link with no parameters, Suprtool executes Suprlink.Pub.Robelle. Suprlink then prompts you for commands, just as it does when you run Suprlink from MPE. When you exit Suprlink, you are returned to Suprtool.

```
>link                                          {invoke Suprlink}
SUPRLINK/Copyright Robelle Solutions Technology Inc. 1988-2001
+input invoices by cust-no                     {Suprlink prompts for commands}
+link  customer
+output invcust
+exit                                          {do Suprlink task and return to Suprtool}
>
```

Passing Commands to Suprlink

You can send commands directly to Suprlink by specifying the Suprlink command as part of the Suprtool Link command. Suprlink executes the command and returns to Suprtool. Suprlink does not execute its task until it receives an Xeq or Exit command.

```
>link input invoices by cust-no
>link link  customer                          {send "link customer" to Suprlink}
>link output invcust
>link exit                                    {exit from Suprlink (not Suprtool)}
```

Notes

You can use the Suprtool2 interface and the Link command to "call" Suprlink. See the Suprcall User Manual for a complete example of how to use the interface to pass commands to Suprlink.

List Command [L]

The List command is used to produce formatted listings of the selected records. You may specify the List command, or the Output command, or both, or neither. If List is used instead of Output, Suprtool sets the Output to $null, so that only a listing is produced.

LIST [OCTAL|HEX|DECIMAL] [CHAR] [NOREC] [LABELS]
[RECORD] [DUPLEX] [ONEPERLINE] [LP]
[NONAME] [NOSKIP] [STANDARD] [DEVICE *name*]
[DATE *format*] [TIME *format*] [PCL *format*]
[LEFTJUSTNUM] [RIGHTJUSTNUM]
[TITLE "*string*"] [HEADING "*string*" ["*string*" ...]]

(Default: Octal/Char or "Formatted")

If Suprtool knows about the fields in the input source (e.g., because you have used the Extract command), the list records are formatted with field names, and internal binary data-types (e.g., integer) are converted to ASCII. You cannot combine the Ask or Query,Num output-options with the List command.

Here is a typical use of List: to find any entries in the m-customer dataset that do not have a valid value for cust-status.

```
>base store,5,READER                    {input from a database}
>get m-customer                         {read this dataset}
>if cust-status<>10,20,30,40            {the only valid values}
>list                                   {print bad entries}
>xeq
```

Format

You can override the defaults with a specification in the List command (e.g., List Hex,Char). If the input source is not self-describing and no Extract command is specified, the default output format is **Octal,Char**, which also shows both input and output record numbers.

Decimal Places

The List command formats numbers using the implied number of decimal places. For example, the following Suprtool commands format the unit cost with two decimal points. We specify the **Rightjustnum** keyword, because numbers with decimal points are hard to read if they are left justified:

```
>item cost,decimal,2                    {two implied decimal points}
>list rightjustnum                      {numbers right justified}
>xeq                                    {  with decimal points}
```

Listing Record Numbers

The **Norec** keyword prevents the printing of the input and output record numbers. The input record numbers are not printed if Output xxx,Data is used, and the file is sorted.

Listing User Labels

The **Labels** keyword causes user labels to be dumped in the format specified. The default format is Octal,Char. If Set Userlabels is Off, the user labels are not printed.

Listing One Field per Line

Suprtool normally attempts to list more than one field on every line of list output. The **Oneperline** keyword causes every field to be shown on a different line.

Listing without Field Names (Noname)

When Suprtool knows the record structure of the output file, it shows the name of each output field. The **Noname** keyword causes the field names to be suppressed. By only extracting a few fields, it is possible to fit the listed output for each record on one line.

Suppressing Blank Lines Between Records

By default, Suprtool prints a blank line between each record. The **Noskip** keyword removes this blank line. If you combine the Noskip, Norec, Noname, and Title options when extracting a few fields, Suprtool can produce a simple report.

Numeric Justification (Leftjustnum and Rightjustnum)

The List command normally left justifies all numeric fields. Specifying List Standard causes all numeric fields to be right-justified, unless you override the default with the Leftjustnum keyword.

Use the **Leftjustnum** or **Rightjustnum** keyword to specify the alignment of the numbers. The two keywords are mutually exclusive. The last one that appears on the command line is the one that is applied.

LaserJet Listings

There are two methods to select different printing options for a LaserJet and other PCL-compatible printers. You can permanently set the PCL option for all listings by using Set List PCL, or you can use the List command to select the PCL option for just one task. PCL stands for Printer Command Language, which is an HP standard for printers. The following is a summary of the PCL values:

PCL	Font	Orientation	Dimensions
1	Lineprinter	landscape	175 cols/60 lines
2	Courier	landscape	100 cols/45 lines
3	Courier "standard"	portrait	80 cols/60 lines
4	Lineprinter	portrait	132 cols/80 lines
5	Courier A4 "tight"	portrait	80 cols/60 lines
6	Lineprinter legal	landscape	223 cols/60 lines

See the Set command for a complete description of the PCL options. By default, Suprtool assumes that the List output device is not PCL-compatible (List PCL 0).

If you use the List command to your terminal with a global Set List PCL value other than zero, your terminal screen may be cleared. To avoid this situation, you can explicitly specify the PCL setting along with the device:

```
>get d-sales
>list serialp,pcl 2
>xeq
```

A4-Size Paper

Most of the PCL options, with the exception of PCL 5, were designed and tested with North American letter-size paper. This is especially true with PCL 5 for A4 paper: it reduces the horizontal spacing between characters so that 80 columns of Courier output fits on a single line. In addition, if you add 2000 to a PCL code, Suprtool adjusts the number of rows and columns for that option to match A4 paper. For example, to print Landscape on A4 paper, use PCL 2001 instead of PCL 1.

In general, selecting A4 paper gives you more space along the long side of the paper and less space along the short side. If you are happy with the way letter-size rows and columns work on A4 paper, simply do not add 2000 to the PCL code. If for some reason you want more columns than provided by Suprtool (e.g., you are using a PCL-compatible system printer with wider paper), you can override the size by using a :File command with the Rec= parameter, as in:

```
:file suprlist;dev=syslp;rec=-220
```

Roman-8 vs. ASCII

The PCL option requests a Roman-8 character set, but some combination font cartridges only supply the ASCII character set (half as many characters means twice as many fonts in a single cartridge). If you ask for Landscape Lineprinter and get Landscape Courier instead, your Lineprinter font probably has the ASCII character set instead of the Roman-8 character set. To request an ASCII font, add 1000 to the PCL code. For example, if you have a Super Cartridge (55 fonts in one!), use PCL 1001, 1004, and 1006. To select both ASCII and A4 paper, add 3000.

Double-Sided Printing on LaserJets

The LaserJet IID and IIID can print on both sides of the paper. The **Duplex** keyword enables double-sided printing on these printers.

```
>list duplex
```

Headings in Listings

Specifying a **Title** in the List command forces Suprtool to produce a formatted listing with page-headings, page-numbers, today's date and the current time. If you want just the date and page numbers, use an empty string. For example,

```
>list title " "
```

The following example prints a report on a LaserJet in Landscape (sideways) mode, using the tiny Lineprinter font, including a page heading with the title. The physical command line limit is 256 characters. As a result, the maximum size of the heading is less than 256 characters because the List command and heading options need to be included in the command line.

```
>in custs                                    {self-describing file}
>if status<>10,20,30,40                      {the only valid values}
>set list pcl 1                              {select LaserJet option}
>list title "Invalid CUSTOMER Records"
>xeq                                         {include title on listing}
```

Changing the Date Format

When you select page headings by specifying a title, each page includes today's date. By default, this date is formatted as mmm dd, ccyy (e.g., Mar 20, 2000). You can override this format with the **Date** keyword. Use the Set command to specify a different default date format for future List commands (e.g., Set List Date 2). The valid date formats are as follows:

Value	Format	Example
0 (default)	mmm dd, ccyy	Mar 20, 2000
1	yy/mm/dd	00/03/20
2	mm/dd/yy	03/20/00
3	dd/mm/yy	20/03/00
4	dd mmmyy	20 Mar00

```
>list title "Example Report" date 3
>xeq                                        {heading date is in dd/mm/yy format}
```

Changing the Time Format

When you select page headings by specifying a title, each page includes the current time. By default, the time is in 24-hour format (e.g., 23:02). You can override this format with the **Time** keyword. Use the Set command to specify a different default time format for future List commands (e.g., Set List Time 2). The valid time formats are as follows:

Value	Format	Example
0	none	
1 (default)	24-hour	23:02
2	AM/PM	11:02PM

```
>list title "Example Report" time 2
>xeq                                        {time will be in AM/PM format}
```

Simple Reports

A Fast Method for Producing Simple Reports

For self-describing files and datasets, the **Standard** keyword is equivalent to List Noname,Noskip,Norec,Rightjustnum with default column headings. For data files, the Standard keyword is equivalent to List Octal,Char. In either case, the Standard keyword provides a default title that describes the input source. You can override the title, date format, time format, or any other option selected by the Standard keyword, by specifying them in addition to the Standard keyword. For example,

```
>get  m-customer
>list standard                              {use all Suprtool defaults}
>xeq

>get  m-customer
>list standard,date 3                       {override the date format}
>xeq

>input mpefile
>list  standard,char                        {override the format options}
>xeq

>input mpefile
>list  standard,leftjustnum                 {left justify numbers}
>xeq

>get  m-customer
>list standard,title "Customer List"        {override title}
>xeq

>get  m-customer
>list standard,heading " "                  {no column headings}
>xeq
```

Listings with Subheadings

When using the Title or Standard keywords, you can also include subheadings with the **Heading** keyword. You can specify multiple columns by repeating the string after the Heading option (e.g., List Heading "First " "Second") or specify the Heading option multiple times (e.g., List Heading "First ", Heading "Second"). Being able to specify multiple columns makes it easier to align column headings when using the Standard keyword. If you specify the Heading keyword without the Title keyword, a default title is produced.

```
>get m-customer                             {read this dataset}
>extract cust-account                       {Z8}
>extract name-first                         {X10}
>extract name-last                          {X16}
>extract credit-rating                      {J2}
>extract cust-status                        {X2}
>sort name-last
>sort name-first
>list standard,title "Customer List", &
      heading "Account  " " First and Last Names" &
              "         " "Credit   " "Status"
>xeq
```

List Device

By default, the List command lists lines to $stdlist. There are several methods that you can use to redirect the output to a specific logical device or to a disc file. You can also redirect output to an attached printer.

File Suprlist

Suprtool lists lines to the file Suprlist. You can use a :File command for the Suprlist file to direct the list output to any line printer (or even to a disc file). The Suprlist file can have widths of 56 to 256 characters. If PCL is zero, Suprlist defaults to 78 columns wide. If your line printer paper is wider, specify the width using Rec= on the :File command.

```
>get m-customer
>:file suprlist;dev=lp;rec=-132          {list output to LP}
>list
>xeq
```

If PCL is non-zero, there is no need for Rec= on the :File command. Suprtool supplies the correct value.

Device LP

Use the **LP** option to send listings to the device "LP". If you have specified a :File command for Suprlist, it overrides the LP option. The LP option assumes that the list device is 132 columns wide.

```
>list lp
```

User Specified Device

Use the **Device** option to specify a specific logical device for the listing. The device name must appear after the Device option and may be a name or number. You may specify a remote device by preceding the remote device name with the system name and a pound sign (#). The Device option assumes that the list device is 132 columns wide.

```
>list device LP                          {same as List LP}
>list device serialp                     {send to device serialp}
>list device Goofy#LP                    {remote device LP on Goofy}
>list device Disc                        {create disc file "Suprlist"}
```

When redirecting the Suprlist file to disc, you must remember to add a disc value to the file equation because the default disc file size for the Suprlist file is 1,023 records.

```
>input sales
>:file suprlist=salefile;disc=20000
>list dev disc
>xeq
```

Listing to Attached Printer

If you wish to list to a printer that is attached to your terminal, use List **Record**. Suprtool uses Record mode on your terminal or PC to print on the attached printer. To use this option, you must be using an HP terminal or HP terminal emulator and your data communication settings must be setup correctly. This option opens a file named Suprcrt instead of Suprlist.

You can combine this option with other listing options. You cannot interrupt Record mode with Control-Y, but you can do a Soft Reset. This unlocks the keyboard and causes the rest of the output to appear on the screen. You can then stop it with Control-Y.

Notes

The List operation occurs logically after the sort phase, if any, and after any Extract, but before the final Output or Put operation. A Reset turns off the current List options.

For more examples of the List command, see *Suprtool Issues and Solutions*.

Listredo Command [LISTREDO]

The Listredo command displays any of the previous 1000 commands.

LISTREDO [*start* [/ *stop*]] [;ABS] [;OUT=*file*]
[*string*] [;REL]
[ALL | @] [;UNN]

(Default: display previous 20 commands)

(BJ and ,, are short for LISTREDO)

Commands are numbered sequentially from 1 as entered and the last 1000 are retained. You can display a single command, a range of commands, all 1000, or all the commands whose name matches the string. You can print the commands with ABSolute line numbers (the default), RELative line numbers (-5/-4), or UNNumbered. You can write the commands to your terminal or Out to a temporary file. If you want to redo any of these commands, see Do, Redo, and Before.

Examples

```
>listredo 5
>listredo 5/10
>listredo help                          {print all Help commands}
>listredo -10                           {print last ten commands}
>listredo ALL                           {print entire redo stack}
>listredo purge                         {print all Purge commands}
>listredo purge xx                      {print all "purge xx" commands}
>listredo @purge                        {print all with "purge" anywhere}
>listredo @;rel                         {print ALL, relative numbers}
>listredo 1/10;out=*lp                  {dump commands to printer}
>listredo @;unn;out=save                {write commands to a file}
```

Notes

The :Listredo command can be abbreviated to BJ as in Qedit, or to ,, (comma comma) as in MPEX. You cannot use a semi-colon (;) to combine commands on the same line.

Persistent Redo

Redo commands can be saved in a permanent file and can therefore be used from another session. You can use the **Set redo** command to specify a filename to save your redo commands. Please see the Set Redo command ("Redo" on page 267) for details.

Numrecs Command [N]

Specify the size of the input file (for tape files), output file (for datasets), and the sort scratch file.

NUMRECS *size* | *percentage*%

(default: *size*=10,000 or EOF of input source)

Parameters

When you know that you are selecting a small subset of the entire input source, use the Numrecs command to reduce the size of the sort scratch file and, if required, the size of the output file. If you select more than *size* entries, Suprtool prints a warning message, and ignores the rest of the input records. However, the output file will have the records that were selected. Use a percent sign (%) to specify the Numrecs as a percentage of the input file size. The percentage can range from 1 to 500.

Using Numrecs for Tape Files

The Numrecs command is required to read more than 10,000 records from a tape file. Suppose you want to read a tape file with 30,000 records. You would use the Numrecs command to increase the size of the output file:

```
>:file t;dev=tape                {define tape file}
>input *t                        {specify Suprtool input}
>numrecs 30000                   {make room for 30000 records}
>output out2                     {output file will have room for}
>xeq                             {   30000 records}
```

Reducing File Sizes

Suppose that the d-sales dataset contains 100,000 entries, but you use the If command to select 15% of the entries. We would specify 15 as the *percentage* on the Numrecs command to reduce the size of the sort scratch file and the output file:

```
>get d-sales                     {specify input}
>numrecs 15%                     {specify 15000 as file size}
>if sales-qty<100                {select a subset of d-sales}
>sort cust-account               {sort using the dataset path}
>output out2                     {output file will have room for}
>xeq                             {    15000 records}
```

MPE Files vs. Datasets

When you specify a source of records using the Input command (as opposed to reading a dataset using the Get or Chain command), Suprtool attempts to duplicate all of the input file's attributes in the output file. This includes the file limit. For this reason, the Numrecs value is ignored if the output file limit is smaller than the input file limit. Numrecs is still useful when reading MPE files to reduce the size of the sort scratch file.

Appending to Files

When you want to create a file that you can append to later, you need to use both the Numrecs command and the Set Squeeze Off command.

```
>get d-sales
>numrecs 200%                          {output file will have extra room}
>set squeeze off                       {reserve room between EOF and LIMIT}
>output dsale
>xeq
...
>get d-history
>output dsale,append                   {dsale has room for appending}
>xeq
```

Open Command [OP]

Specify an SQL database to open. Only one database can be open at a time.

OPEN ALLBASE *dbename owner*

Allbase Database

Use the Open command to connect to an Allbase database. You must specify your Allbase *dbename* and *owner*. To read data from an Allbase table or view, use the Select command. Use Set Allbase Rows to specify how many rows for Suprtool to fetch when it is reading Allbase data.

```
>open allbase inventory anne
```

Output Command [O]

Define the name of the output file as one of these: a new disc file (default), an existing MPE or KSAM file (Append or Erase option), * for $stdlist, a back reference to a :File command (*name), or =Input to sort a file into itself. If you use List, Put, Update or Total, Output defaults to $null. Output produces the same record format as the input source (adjusted by Extract commands), unless you override it with format keywords.

OUTPUT *filename format* [TEMP] [ERASE | APPEND]

(Default: DATA only, "new" file)

New Files

The *filename* is the name of a new MPE disc file to be built by Suprtool (Output xxx). Unless you use a :File command to specify explicitly the record length and blocking factor of the new file, Suprtool calculates the record length itself and chooses a blocking factor to optimize batch access to the new file.

If Suprtool cannot Save the new file because of a duplicate file name, you may purge the old file or give the new file a different name. If Suprtool is running in batch mode, it renames the new file as Output*nn*, where *nn* is the lowest number between 00 and 20 that makes a valid new file name. The Output*nn* file is built in the same group and account as the duplicate output file and not in your logon group and account. This prevents problems on private volumes.

In many cases, Suprtool creates the end-of-file and the file-limit the same. If you intend to append records to the output file, use Set Squeeze Off to force Suprtool not to reduce the file-limit when closing the output file.

Suprtool does not create new KSAM files. Use KSAMUTIL or the MPE/iX :Build command to create the KSAM file, then use Suprtool to load it as an existing file.

Temporary Files

By default, Suprtool saves the output file as a permanent file. Use the **Temp** option to force Suprtool to save the output file as a temporary file. If the output file already exists as a temporary file, Suprtool prompts you to purge the existing temporary file. If you do not want to purge the old temporary file, Suprtool prompts for a new temporary file name. In batch, Suprtool renames the output file as the temporary file Output*nn*.

Permanent and Temporary Files

If you have a temporary file and a permanent file of the same name, Suprtool will attempt to create a new permanent file.

If your Output command specifies the append or erase options Suprtool will attempt to output to the temporary file.

Existing Files

The *filename* is the name of an old disc or KSAM file to erase (Output xxx,erase) or to append to (Output xxx,append). Output xxx,erase does not purge the existing file; it simply overwrites the contents of the file. The original attributes of the file such as record size, blocking factor, and file limit are not changed.

If the old file is a KSAM file, Suprtool attempts to add the output records to the KSAM tree structure. If duplicate keys are not allowed in the KSAM file, there may

be FWRITE errors due to duplicates. The treatment of these is controlled by the Set Ignore command. By default, a duplicate error causes the extract task to terminate.

Tape or Line Printer Files

The *filename* is a back reference to a magnetic tape or line printer file via a :File command (Output *XX). When sending output to magnetic tape, you should specify the record size and blocking factor on the :File command. Use the same :File command (with the same record size and blocking factor) when reading the tape file.

$Stdlist

Specify output to $stdlist by a single asterisk (Output *). Use this *filename* with the ASCII option for a quick listing of a file or dataset.

Overwriting the Original File

Use the special *filename*=Input to specify erasing the unsorted records by writing the sorted records back into the original input file. Disc space is saved because a new output file is not required, but this option cannot be restarted! The =Input option can only be used for sorts to disc files and KSAM files, not to IMAGE datasets. You can use this feature to reorganize existing KSAM files by extracting the records you wish to retain, sorting them into your optimal key sequence, and writing them back into the same KSAM file.

```
>in workfeb
>key 1,10
>output=input
```

Note that typing "input" after the equal sign is optional. Any name typed after the equal sign is ignored; it is the equal sign itself that directs Suprtool to overwrite the input file.

Size of the Output File

Suprtool builds new output files with a capacity large enough to hold all the input records. You can determine the capacity (limit) in two ways:

1. You can make the capacity larger by specifying a Numrecs command with either a specific value or a percentage greater than 100.

   ```
   >numrecs 150%
   ```

   For IMAGE datasets, you can also reduce the size of the output file using Numrecs. When the input is not an IMAGE dataset and you select a Numrecs smaller than the number of records in the input file, Suprtool builds the output file the same size as the input file. However, it limits the number of records written to Numrecs.

2. You can use a :File command to override Suprtool's choice.

   ```
   >:file sales; disc=100000
   >output sales
   ```

The file is built by Suprtool as it begins the execution phase (when you type Xeq or Exit). If there is not enough disc space you get a file system error 46. Use a :File command to choose a smaller file size.

At the end of execution, Suprtool usually reduces the capacity (limit) to the actual number of records in the output file (EOF) which in turn reduces the disc space. If you do not want this to happen, use the **Set Squeeze Off** command.

Format Options

The format and length of the output records is determined by which of the following *format* keywords is selected:

Keyword(s)	Format
DATA	Default
ELSE	Records not qualified by if condition
KEY	Sort keys only
NUM	J2 record numbers only
NUM,KEY	J2 record numbers plus sort keys
NUM,DATA	J2 record numbers plus data record
QUERY	Self-describing file
LINK	New format self-describing file
NUM,QUERY	Query "numbers" format
ASK	COGELOG ASK select file
ASCII	Convert numeric to ASCII
DISPLAY	Convert numeric to display (zoned-decimal)
PRN	Personal computer format

DATA

DATA is the default; output records are the same length and format as the input records.

ELSE

ELSE tells the output command to output any records to the temporary ELSE file that do not qualify under a certain if command. If you have a task that looks for all records with an amount >= 0, then the records <0 will be written to a temporary file called ELSE. This file can be renamed with a file equation. The ELSE option cannot be used with num or key or any duplicate option or command or the Sort command. The ELSE file that gets created will look like the output file in terms of attributes.

KEY

KEY creates output records containing only the sort keys, concatenated from left to right (major key to minor key).

NUM

NUM creates output records of four bytes in length, containing the double-integer record number of each input record. The record number of the first record is 1 for IMAGE datasets, 0 for MPE files, and either 0 or 1 for KSAM files, depending on whether the FIRST=0 option was selected when the KSAM file was built. The NUM option is not recommended for IMAGE master datasets because entries in master datasets can migrate. Detail datasets and MPE and KSAM files are okay.

NUM,KEY

NUM,KEY creates output records containing the four-byte record number, followed by the sort key values, concatenated from left to right (major key to minor key).

NUM,DATA

NUM,DATA creates output records containing the four-byte record number, followed by the full data record. This can be useful to create "transaction" files from detail datasets that will later be updated back to the database after a processing stage.

QUERY

QUERY creates an MPE self-describing output file. A self-describing file has a file code of "SD" in the :Listf command and an actual file code of 1084. Self-describing files contain information about the structure of each record in the file. The file label contains the name, type, length, and offset of each field in the file.

Self-describing files have no provision for repeated fields. These fields appear with an "unknown" type in the file label. The QUERY option produces a file that is similar to the file produced by the SAVE command of QUERY.

LINK

The QUERY output option has some major drawbacks. Compound fields are not handled, date and decimal point information is not saved, and sort information is not part of the file description. The LINK option produces a self-describing file that solves all these problems. This option is the recommended way to generate files for Suprlink. We are also encouraging third-party software vendors to accept this format.

To convert a self-describing file back into a non-SD file, see "Suprtool and Self-Describing Files" on page 71.

NUM,QUERY

NUM,QUERY creates output records in QUERY "numbers" format. This is an undocumented feature of QUERY that is used by some application packages. Records in a "numbers" file contain the dataset number and the record number in ASCII format. A QUERY "numbers" file is usually used as input to a report program. Substituting Suprtool for QUERY in these batch jobs should improve the speed.

ASK

ASK is a QUERY-replacement tool from COGELOG. ASK creates output records in ASK select-file format. Suprtool can be used to scan and select records quickly from a dataset. ASK Version C can produce reports from the resulting select-file. This option cannot be used with any other Output option.

Recent versions of ASK read self-describing files. For these versions use the Link option instead of the Ask option.

ASCII

ASCII converts all binary input fields into their equivalent ASCII values. All binary fields are right-justified in their fields with a trailing sign. The sign can be a blank, "+", or "-". Byte fields are not affected by this option. The size of the ASCII field depends on the format of the binary field:

Field Format	Output Size

I1, J1	6 bytes
I2, J2	11 bytes
I3, J3	16 bytes
I4, J4	20 bytes
K1	5 bytes
K2	10 bytes
E2	12 bytes
E4	23 bytes
R2	12 bytes
R4	23 bytes
Z*n*	*n*+1 bytes
P*n*	*n* bytes

Any fields with implied decimal places (see the Item command) are formatted with a decimal point in the correct position. Suprtool reserves two additional positions for each output field that has an implied decimal point.

DISPLAY

DISPLAY converts all binary input fields into their equivalent display values (also known as zoned-decimal). The size of the display field depends on the format of the binary field:

Field Format	Output Size
I1, J1	5 bytes
I2, J2	10 bytes
I3, J3	15 bytes
I4, J4	19 bytes
K1	5 bytes
K2	10 bytes
P*n*	*n*-1 bytes
P**n*	*n*-2 bytes

The last digit of the number is replaced with a letter corresponding to positive or negative values. See the following table.

Units Digit	Positive	Negative	No Sign
0	{	}	0
1	A	J	1
2	B	K	2
3	C	L	3
4	D	M	4
5	E	N	5
6	F	O	6
7	G	P	7
8	H	Q	8
9	I	R	9

Positive values for I- and J-type fields are converted into neutral (i.e., no sign) display values. The sign is preserved when converting packed fields to display.

DISPLAY is not supported for Real or IEEE fields. Byte fields are not affected by this option. Display fields in the input record are not converted into display.

PRN

Many PC software packages import PRN files (these files are also called delimited in some PC documentation). A PRN file has quotes around character-fields and a comma between each field. Binary values are output in ASCII with an optional leading sign. Not all applications accept PRN files; for more precise conversion of data, use STExport.

PRN converts each input-field to a fixed-width output-field, filling with trailing spaces where necessary. All binary values are converted into their equivalent ASCII value, left-justified in their fields. The sign precedes the ASCII value for the number and can be missing, "+", or "-". See the ASCII output-option for the field width of each data-type.

Output fields with implied decimal places (see "Decimal Places" on page 217) are formatted with a decimal point in the correct position. Like the ASCII option, Suprtool reserves two extra columns for each output field with implied decimal places.

Lotus 1-2-3 accepts records only up to 240 bytes long. Because the PRN option leaves room for the maximum value of any field, you may need to restrict the number of output fields using the Extract command.

While some PC software allows alternate characters to be used to delimit character fields, Lotus 1-2-3 accepts double quotes only. Since Lotus 1-2-3 rejects character fields that contains a double quote, Suprtool *removes* all double quotes from character fields when generating the PRN format. Suprtool removes quotes by replacing them with a space.

See STExport for a method of including header lines in the file to be down-loaded.

Examples

One reason to use $stdlist as the output file (indicated by * as the filename) is to obtain a quick listing of the ASCII fields in the input source. The following example lists the cust-account, name-last, and name-first fields of the m-customer dataset and separates them by two spaces:

```
>base store,5,READER                {input from a database}
>get m-customer                     {use the m-customer dataset}
>extract cust-account               {account number will be first}
>extract "  "                       {two spaces}
>extract name-last                  {the customer's last name}
>extract "  "                       {two more spaces}
>extract name-first                 {the customer's first name}
>output *                           {output the records to $stdlist}
>sort cust-account                  {sorted by the dataset's key field}
>exit
```

The following examples demonstrate other combinations of options on the Output command. The entire *Issues* chapter of the manual should be reviewed for extended examples using the Output command. Many Output options were intended for specific application areas.

```
>output newfile
>:file dtape;dev=tape;rec=40,20
>output *dtape
>output ksamfile,key,erase
>output accum,append
>output keyfile,key
>output transf,num,data
>output querynum,num,query
>output $null
```

If you want to find out how many records are qualified by some selection criteria, without producing an output file, send the output to $null. The Out= count on $stdlist, and the JCW SuprtoolOutCount are both set to the number of qualifying records.

ELSE

While Suprtool does not have traditional else logic in its if command you can output all of the records "discarded" for a given task out to a temporary filed named ELSE.

This way you can output multiple files for a given task as in:

Notes

The output file *must* be an "old", existing file if the Append or Erase option has been specified. If you use a :File command to force Output to an old file without specifying one of these options, Suprtool rejects the file.

If the record length of the output file is not the same as expected by Suprtool, the records are automatically truncated or expanded with either binary zeros or blanks. If the blocking factor is not as expected, Suprtool automatically "reblocks" the records. Thus, Suprtool can be used to reblock disc or tape files economically into better block sizes in order to save disc space, or to improve on-line access, or to speed batch access.

Put Command [P]

Directs output records to be DBPUT into an IMAGE dataset. Put can be combined with Output to a file, but if no Output command is entered, Suprtool defaults the output file to $null. It is your responsibility to ensure that the output records match the format of the target dataset. The DATA option of the Output command is the only valid output option when Put is specified.

PUT *setname* [,*database* [,*mode* [,*password*]]]

(Default: *base*=Base, *mode*=1, *pass*=Creator)

Parameters

If *database* is not specified, *setname* must be a valid dataset of the Base previously opened. You must have write access to the dataset selected.

If *database* is included, the specified database is opened. The specified database can be redefined to reside on another system or in a different account/group via a :File command. The default open mode is 1 and the default password is Creator. This Put database is closed after the task is completed, unlike the database opened with Base. It is possible to copy records from one database to another with Suprtool.

In session mode, Suprtool prompts for the *password* if it is not included in the command. In batch mode, it is necessary to specify a database password without the password being echoed on the job stream listing. A special database password is provided to allow for this. When a question mark "?" is used as the database password, Suprtool prompts for the password on the next physical input line without echoing. This occurs in batch mode or in session mode.

Examples

For the first example, we assume that the D-SALES dataset is designed to contain a week's worth of transactions. Each night, the transactions from a week ago are moved to the monthly dataset: D-SALES-MON. The D-SALES and D-SALES-MON datasets have exactly the same field declarations. The following Suprtool task would move one day's transactions from D-SALES to D-SALES-MON.

```
>base store,1,write                {requires write access to database}
>get d-sales                       {read the weekly dataset}
>if purge-date=000401              {select one day's transactions}
>put d-sales-mon                   {adding them to the monthly dataset}
>exit
```

For the next example, we assume that the D-SALES-MON dataset is in a different database. We prompt for the password for both databases:

```
>base store,1,?                    {Suprtool must prompt for password}
Password >                         {password not echoed}
>get d-sales
>if purch-date=000401              {select one day's transactions}
>put d-sales-mon,ostore,1,?        {add to OSTORE database}
Password >                         {password not echoed}
>exit                              {the output file defaults to $null}
```

This example shows how Suprtool can read an MPE disc file and Put each record in a database:

```
>input dsales                          {"dsales" has exactly the same format}
>put d-sales,store                     {   as the d-sales dataset}
>exit
```

Improving Put Performance with Deferred Output

Suprtool uses the standard Image intrinsic DBPut to add entries from a dataset. This makes tasks with a Put operation slower than the standard MR/NOBUF reads, but insures structural integrity of the database and that any Third-Party Indexes are kept in sync.

You can reduce the time needed to Put dataset entries by enabling deferred output for the Put database. When Set Defer is On and the Put database is opened in mode-3 (exclusive), Suprtool calls DBCONTROL to notify IMAGE that disc writes are to be deferred whenever possible. The database creator can enable deferred output processing for all users of a database with DBUTIL (enable base for autodefer).

Deferred output can improve the speed of DBPUT significantly, but it has a major drawback: if the system crashes (i.e., you have a system failure or a nonrecoverable power failure) while you are in deferred output mode, you must RESTORE your database from tape. If Suprtool stops due to an error, or if you abort Suprtool (i.e., break/abort or :abortjob) while the database is opened in deferred output mode, your database does not have to be restored.

Omnidex

If you are adding records to a dataset with Omnidex keys, you must have MPE/iX third-party indexing enabled in your database, or you must run Suprtool with Omnidex's call conversion facility.

```
:run suprtool.pub.robelle;xl="xl.pub.disc"
>base store
>input  mpefile
>put    d-sales                       {d-sales has Omnidex keys}
>exit
```

Notes

If the dataset selected for output is a master dataset, it is possible that some of the output records may be duplicate key values. If this happens, Suprtool prints an error message and terminates the task, unless you have enabled Set Ignore, in which case the duplicates are counted and reported at the end of the task. Duplicate records are not written to the output file.

To add individual entries into a dataset, specifying each field value in ASCII, use the #Add command of Dbedit.

Q Command [Q]

Prints a message on $stdlist.

Q [*"string"*]

(Default: print a blank line)

The *string* of up to 253 characters is printed on $stdlist. The string is truncated to the record width of $stdlist. In many cases, job streams are limited to 80 or 132 characters.

The string must be embedded in quotes. Either single-quotes (') or double-quotes (") are permitted. The quotes are not printed on $stdlist.

Use the Q command in usefiles for printing explanations. Put :COMMENT in usefiles for a nonprinting comment line, then invoke the file with the USEQ command.

Examples

The Q command is often combined with the Userpause command. The example is a usefile that provides an explanation of how long a task takes:

```
>q
>q "We will select all transactions over $10,000.  Since"
>q "there are many transactions, this task will take"
>q "some time (usually more than fifteen minutes)."
>q
>userpause "Press any key when you are ready to start."
```

Redo Command [REDO]

Enables you to modify and repeat any of the previous 1000 command lines.

REDO [*start* [/ *stop*]]
[*string*]
[ALL | @]

(Default: redo the previous command)

(Comma "," is short for REDO)

The Redo command allows you to modify the commands before it executes them. If you don't need to change them, use the Do command. Commands are numbered sequentially from 1 as entered and the last 1000 are retained. Use the :Listredo command to display the previous commands. You can redo a single command, a range of commands, or the most recent command whose name matches a string.

The Redo command uses MPE-style editing logic (D, I, R, U and >). The default mode is to replace characters. To delete, type DDDD under the characters to be removed. To insert, type I under the insertion spot, then the new characters. To undo your changes, type U. To append to the end of the line, use >xxx. To delete from the end of the line, use >DD. To replace at the end of the line, use >Rxxx. And to erase the rest of the line, use D>. If you prefer Qedit-style editing (Control-D, etc.), use the Before command instead of the Redo command.

Examples

```
>listf @.soruce                          {"source" is not spelled right}
NON-EXISTENT GROUP.  (CIERR 908)
>redo                                    {redo most recent command}
listf @.soruce                           {last command is printed}
         our                             {you enter changes to it}
listf @.source                           {edited command is shown}
                                         {you press Return}

>listredo all
>redo 5                                  {redo 5th command in stack}
>redo                                    {redo previous command}
>redo -2                                 {redo command before previous}
>redo 8/10                               {redo 8th through 10th}
>redo -10/                               {redo -10 through last}
>redo purge                              {redo last Purge command}
>redo purge temp                         {redo last "purge temp"}
>redo @temp                              {redo last containing "temp"}
```

Notes

The Redo command can be abbreviated to a comma, as in MPEX. You cannot use ";" to combine commands on the same line. The Suprredo file is always a temporary file. To save more commands, use a :File command on the Suprredo file before you run Suprtool:

```
:file suprredo;disc=5000
:run suprtool.pub.robelle
```

Hpmodify Keys – Reference

Here are the MPE-style REDO sub-commands:

Directive	Effect
i	INSERT. If text follows the i, this text is inserted in the current line starting at the i position.
r	REPLACE. If text follows the r, this text replaces the same number of characters in the current line beginning at the r position.
d	DELETE. Deletes a character from the current line for each d specified in the edit line. Note that "d d" does not specify a range as it does in MPE V but simply deletes one character above each d. Multiple d's may be followed by an INSERT or REPLACE operation.
d>	DELETE. Deletes to the end of the current line from the position specified by d>. May be followed by an INSERT or REPLACE operation.
>	APPEND. If text follows the >, this text is appended to the end of the current line. If a > without text is positioned beyond the end of the current line, then a simple replacement is performed instead.
>d	DELETE. Deletes from the end of the current line, right-to-left. Multiple d's and INSERT and REPLACE strings may be specified after >.
>r	REPLACE. Replaces characters at the end of the command line. The last (rightmost) character of the replacement string is at the end of the line.
c	CHANGE. Changes all occurrences of one string to another in the current line starting at the c. The search string and replace string must be properly delimited. A proper delimiter is a non-alphabetic character (such as ' " or /). The substitution is specified as c*delim search-string delim* [*replace-string* [*delim*]]. Omitting the *replace-string* causes occurrences of *search-string* to be deleted, with no substitution.
u	UNDO. A single u in column one cancels the most recent edit of the current line. Using the Undo command twice in a row cancels all edits for the current line and re-establishes the original, unedited line. If u is placed anywhere other than column one of the current line, then a simple replacement is performed. Undo makes sense only if you have a line on which you have performed some editing that can be "undone."
other	Simple replacement. Any other character (not i, r, d, d>, >, >d, >r, c, or u) will be put into the current line at the position above where it is placed, replacing any existing character. Simple replacement also occurs for the editing characters i, r, c, or > if they are not followed by text; or if > appears at or beyond the current end of line.

Hpmodify Examples

Here are examples of the MPE-style REDO sub-commands in action:

Edit	Action
`u`	First occurrence undoes the previous edits. The *u* must be in column one.
`u`	Second occurrence undoes all edits on the current line. The *u* must be in column one.
`rxyz`	Replaces the current text with xyz starting at the position of *r*.
`xyz`	Replaces the current text with xyz starting at the position of *x*.
`ixyz`	Inserts xyz into the current line, starting at the position of the *i*.

`ddd`	Deletes three characters, one above each *d*.
`d xyz`	Deletes a single character above the *d*, skips one space, then replaces the current text with xyz starting at the position of *x*.
`ddixyz`	Deletes two characters, then inserts xyz in the current line starting at the position of the *i*.
`d  d`	Deletes one character above the first *d*, skips two spaces and deletes a second character above the second *d*. It does not delete a range of characters, making it unlike the MPE V version of Redo.
`d  d>xyz`	Deletes a single character above the first *d*, skips two spaces and deletes to the end of the line beginning at the second *d*, and then places xyz at the end of the line.
`>xyz`	Appends xyz to the end of the current line.
`>ddxyz`	Deletes the last two characters from the end of the current line and then places xyz at the end of the line.
`>rxyz`	Replaces the last three characters in the current line with xyz.
`>ixyz`	Appends xyz to the end of the line. In this case, the *i* command is superfluous, because > accomplishes the same result. Using >xyz would be sufficient.
`c/ab/def`	Changes all occurrences of ab to def.
`c"ab"`	Deletes all occurrences of "ab".
`cxyz`	Replaces the current text with cxyz, starting at *c*. Because delimiters have not been specified (as they were in the previous two examples), this is a simple replacement with the four characters.

Persistent Redo

Redo commands can be saved in a permanent file and can therefore be used from another session. You can use the **Set redo** command to specify a filename to save your redo commands. See "Redo" on page 267 for details.

Reset Command [R]

Resets aspects of the current task.

RESET [ALL | @ | *command* [...]]

(Default: Delete/Sort/Key/If/List)

Parameters

More than one command can be Reset at once by entering several *commands* separated by a space or a comma. If no parameters are specified, Suprtool cancels the previous Sort, Key, If, Delete, and List commands. The other commands remain unchanged.

If ALL is specified, all of the Input and Output commands are canceled, and database/files are closed. In fact, the only options that are not reset to the initial condition are Define, Item, and Set options.

Examples

You began to specify a sort, but then you discovered that you specified the wrong database, so you decided to start all over:

```
>base ostore,5,READER                {open for read access only}
>get d-sales
>sort cust-account
>reset all                           {oops, wrong database}
>base store,5,READER                 {now we have correct one}
```

In the next example, you entered an incorrect If command:

```
>if delivered>000401                 {wrong field used}
>reset if                            {only reset the If command}
>if purchased>000401                 {    and specify it again}
```

This time both the If command and the Extract commands are incorrect:

```
>if delivered>000401                 {wrong field used}
>extract delivered                   {  in both commands}
>reset if,extract                    {only reset the If and Extract}
>if purchased>000401                 {  commands and start again}
>extract purchased,account
```

Notes

By resetting certain commands, other commands are also reset. For example, resetting the Base command resets almost all other commands, except the Define and Output commands. Resetting the Put command closes the output database when the database for the Put command is different from the input database. Resetting the Chain, Get, or Input command resets everything except the Base, Define and Table commands. Resetting the Table command resets everything except the Base and Define commands. Resetting either the Define or Item command resets both Define and Item settings.

Select Command [SEL]

Specify a select statement for the currently open SQL database. The select command supports all of the features of the select command of the open SQL database. Only one select command can be specified at a time.

SELECT *select-command*

Allbase sorts data:

```
>select * from user.account@emp order by ename
```

Suprtool sorts data:

```
>select * from user.account@emp
>sort ename
```

The *select-command* can contain any expression or clause that is supported by the SQL database. Some features (e.g., ORDER BY) may be done faster by Suprtool (e.g., the Sort command).

Some Select commands can exceed the 256-character command line limitation. The Select command, however, can be considerably larger if you use the $read feature of the Select command. This feature is similar to the If command $read feature and is invoked by entering the Select command on a line by itself (unlike If, you do not specify $read explicitly).

```
>select
-ename,salary,tax_paid
-from scott.employee
-order by ename
-//
>
```

The Suprtool prompt changes from ">" to "-" after entering the Select command by itself on a line. The entire command gets sent to the Select command parser after terminating with two slash marks (//) or a blank line.

You might realize greater performance gains with the Select command if you specify only the columns that you need in your output file for either tables with many items or when you need only a couple of items from a given table.

The following Select command

```
>select col1,col2,col3 from user.account@emp
```

may be faster than

```
>select * from user.account@emp
```

Your mileage may vary.

Set Command [S]

Enables or disables certain operating options within Suprtool. These options are not reset by Xeq or Reset commands.

SET ALLBASE ROWS *number*

SET ARITHMETIC CLASSIC|IEEE

SET BACKWARDCHAIN ON|OFF

SET BASECLOSE ON|OFF

SET BLOCKSIZE [*size*]

SET BUFFER [*size*]

SET CLEANCHAR <string>

SET CURRENCYSYMBOL <string>

SET DATE CUTOFF [*number*]

SET DATE FORCECENTURY ON|OFF

SET DATE IFYY2000ERROR ON|OFF

SET DATE MAPTOPHDATE8 ON|OFF

SET DECIMALSYMBOL <string>

SET DEFER ON|OFF

SET DUMPONERROR ON|OFF

SET EDITSIGNNEUTRAL ON | OFF

SET EDITSTOPERROR ON | OFF

SET EOFREAD ON|OFF

SET ENDIANINT BE|LE (has no effect in Suprtool/MPE)

SET ENDIANLOG BE|LE (has no effect in Suprtool/MPE)

SET FFISBE ON|OFF (has no effect in Suprtool/MPE)

SET FILECODE *number*

SET FILENAME Help|Link|Edit|Hint|Export|Outcount *filename*

SET FIRSTREC [0|1]

SET HINTS ON|OFF

SET IFCHECK ON|OFF

SET IGNORE ON|OFF

SET ITEMABBREVIATEDATE ON|OFF

SET INTERACTIVE ON|OFF

SET INITNUMNEXTENT 1..32

SET ITEMLOCK <*fieldname*>

SET LABELLEDTAPEREWIND ON|OFF

SET LIMITS [Mpe ON|OFF] [ReadOnly ON|OFF] [Tablesize *size*]

SET LIST DATE *number*

SET LIST PCL [0|1|2|3|4|5|6]

SET LIST TIME *number*

SET LOCK [*number*]

SET MAXNUMEXTENT 1..32

SET NLS [*number*]

SET NUMBUG ON | OFF

SET OPENMODE [*number*]

SET ORACLE ROWS *number* (not supported in Suprtool/MPE)

SET ORACLE INTEGER ON | OFF (not supported in Suprtool/MPE)

SET ORACLE OPENFIX ON|OFF (not supported in Suprtool/MPE)

SET ORACE ZERONULL ON | OFF (not supported in Suprtool/MPE)

SET PATTERN NEW|OLD

SET PREFETCH [*number*]

SET PRIVMODE ON|OFF

SET PROGRESS Percent [*number*] Minimum [*number*]

SET PROMPT *character*

SET REALMAP ON|OFF (has no effect in Suprtool/MPE)

SET RECOVER ON|OFF

SET REDO *filename*

SET SORTEXTMEM ON|OFF

SET SORTFAST ON|OFF

SET SORTMEMADDR *number*

SET SQUEEZE [ON|OFF]

SET STATISTICS ON|OFF

SET SUBSYSTEM ON|OFF

SET SUSPEND ON|OFF

SET THOUSANDSYMBOL <string>

SET USERLABELS ON|OFF

SET VARSUB ON|OFF

SET VARSUBCOMPAT ON|OFF

SET VARSUBDEBUG ON|OFF

SET WARNINGS ON|OFF

SET XLTRIM ON|OFF

Each option is its own Set command. That is, you cannot specify multiple options in the same command. Instead, use multiple Set commands. For example, to Set Squeeze Off and to define a PCL device for the List command, you would specify two Set commands:

```
>set squeeze off
>set list pcl 1
```

or

```
>set squeeze off;set list pcl 1
```

Add system-wide Set commands to the Suprmgr.Pub.Sys configuration file.

When Suprtool starts running, all the options are set to initial "factory settings". The Suprmgr file may contain Set commands to override the initial values. Set commands remain set until the end of the Suprtool run, or until changed by another Set command.

Some commands have optional parameters (e.g., the numeric value of Set Lock). If the command is specified without a value, the default that Suprtool uses may be different from the program's initial value. When that is the case, the command description will show the initial value and the default-when-omitted value.

Allbase

SET ALLBASE ROWS *number*

(Initially: 100)

If you open an Allbase database, Suprtool reads more than one row at a time when processing the input source. By default, Suprtool fetches 100 rows at a time. You can vary the number of rows that Suprtool fetches with Set Allbase Rows. The minimum number of rows is 1 and the maximum number is 990. You must specify Set Allbase Rows before entering the Select command.

Arithmetic

SET ARITHMETIC Classic | IEEE

(Initially: depends)

Specify which data-type Suprtool should use for arithmetic expressions by using Set Arithmetic. The default is Classic for Suprtool/V, and IEEE for Suprtool/iX and Suprtool/UX. This option is only supported in Suprtool/iX.

Baseclose

SET BASECLOSE ON | OFF

(Initially: OFF)

Use Set Baseclose On to ensure that Suprtool closes your database whenever it exits. The default is for Suprtool to hold the database open when it suspends (e.g., when returning to Qedit).

Blocksize

SET BLOCKSIZE [*size***]**

(Initially & Default: <none>)

When Suprtool is copying an MPE or KSAM file, it always attempts to duplicate the record size and blocking factor of the original file. In all other cases, Suprtool chooses an optimal blocking factor, but restricts the total block size to 2048 words. When Suprtool is computing the block size, it always chooses the smallest block size

divisible by 128. Specifying an explicit block size forces Suprtool to pick the largest block size that can fit in the specified block size and still be divisible by 128.

Users may wish to add the Set Blocksize command to the Suprmgr file. Increasing the block size has two potential problems: small files may consume more disc space, and reading the file buffered requires more memory for the MPE file buffers. The maximum block size allowed is 8,192 words.

Buffer

SET BUFFER [*size* **]**

(Initially: depends; Default: no change)

Set Buffer varies the maximum number of words that Suprtool allocates for input and output buffers. This buffer size limits the number of data blocks that Suprtool reads at a time.

Suprtool overrides the maximum buffer size specified, if that is necessary to get the task done. For example, when the blocks size of the file is greater than buffer maximum specified.

The maximum value is 14,336 words in Suprtool/V and 24,576 words in Suprtool/iX. The default value used by Suprtool is the largest size that fits in the available data structures and still allows Suprtool to get the job done. Using Set Buffer to reduce the size of the buffer reduces the size of the stack that Suprtool uses for copy operations. Suprtool/V always uses a buffer size of 4096 words when sorting. This leaves the maximum amount of room in the stack for the sorting operation.

Some utilities that intercept all file system calls can have problems with the large read/write requests that Suprtool makes (e.g., some versions of Omnidex and Netbase versions less than 0.9.4). You can avoid this problem by adding a Set Buffer command to Suprmgr.Pub.Sys to reduce the default buffer size. Suprtool automatically detects when a file is considered Netbase remote and adjusts the buffer size for you. This requires Netbase 0.9.4 version or higher.

CleanChar

SET CleanChar " "

(Default: space)

By default, Suprtool will replace any of the characters specified in the clean command with a space. You can specify what character to use to replace any of the characters that qualify with the following set command:

```
>set CleanChar "."
```

This will set the character to replace any of the qualifying "to be cleaned" characters to be a period.

If you want to Clean a field and not replace with any character you just need to set the clean character in the following manner:

```
>set CleanChar "<null>"
```

CurrencySymbol

SET CurrencySymbol "$"

(Default: Dollar Sign)

This setting is used to change the Currency character which is used to edit the data in a field used by the $Number function. The defined currency symbol(s) is stripped from the field prior to converting to number, in the $number function. The CurrencySymbol can be up to four characters.

```
>set CurrencySymbol "$"
```

The default character for the Currency Symbol is the Dollar sign.

Date Cutoff

SET DATE CUTOFF [*number* **]**

(Initially: 10; Default: 00)

Date Cutoff tells Suprtool what century to use when Suprtool generates the date value from the $date and $stddate functions. This setting only affects the date values generated by the $date and $stddate function in the If and Extract commands. This does not affect user data.

Versions of Suprtool without Set Date Cutoff would assume 19 for the century for any user-specified $date with a two-digit year.

Now with Set Date Cutoff *xx*, Suprtool assumes the following: a value of 20 for the century if the two-digit year specified in the $date or $stddate functions is less than the value of Set Date Cutoff; a value of 19 for the century if the two-digit year is greater than or equal to Set Date Cutoff.

The initial value of Set Date Cutoff is 10. Therefore, the default behavior in $date and $stddate is to treat the two-digit years with values of 00..09 as 2000..2009, and the two-digit years with values of 10..99 as 1910..1999.

yy Value in $date	With Cutoff 10	With Cutoff 25
00	2000	2000
…	…	…
09	2009	2009
10	1910	2010
11	1911	2011
…	…	…
24	1924	2024
25	1925	1925
26	1926	1926
…	…	…
99	1999	1999

We recommend that you always provide a four-digit year when using $date. However, for reasons of backward compatibility, we introduced Set Date Cutoff. See "Date ForceCentury" on page 253 for more information.

$Stddate and Set Date Cutoff

When $stddate has to convert from a date in only a two-digit format, the conversion to the four-digit date will use the value in Set Date Cutoff.

For example,

```
>get sales-detail
>set date cutoff 15
>def new-ship-date,1,4,double
>item ship-date,date,mmddyy
>ext order-no / sales-amount
>ext new-ship-date = $stddate(ship-date)
>out salesinfo,link
>xeq
```

In this example, if any ship-date has a year of 14 or less, then the century applied to the new-ship-date field will be 20. Ship-dates with a year of 15 or more will have a century of 19 applied.

Date ForceCentury

SET DATE FORCECENTURY ON | OFF

(Initially: OFF)

Set Date ForceCentury On will not allow a yy date to be entered in the $date function, it will force the user to enter a full ccyy date.

```
>set date forcecentury on
>item date-field,date,ccyymmdd
>if date-field >= $date(00/12/10)

Error:You must specify the century or Set Date ForceCentury off
```

The default value for Set Date ForceCentury is off.

Date IfYY2000Error

SET DATE IFYY2000ERROR ON | OFF

(Initially: ON)

By default, Suprtool considers dates with a two-digit century component from the $date and $today functions to be invalid when the dates resolve to be greater than 1999 and the If operation is a relative operation (e.g., greater than or equal to). You can control whether Suprtool considers this condition an error by using the following Set command:

```
>set Date Ifyy2000Error Off
```

The following example shows what is considered to be an error by the If command and how the Set command can turn off the error check:

```
>def a,1,6
>item a,date,yymmdd
>if a >= $today
           ^
Error:  Cannot use a date beyond 1999 for this format
>set date ifyy2000error off
```

We have chosen this condition to be an error by default because when the $today or $date function in the If command resolves a date in a yymmdd format to a value beyond 1999, the result is not always a useful value. For example, a December 10, 2000 date in a yymmdd format would have a value of 001210 and comparisons to this value could be logically incorrect.

If you would have included a Delete command in a dataset selection task, you could have removed all of your records.

Date MapToPHDate8

SET DATE MAPTOPHDATE8 ON | OFF

(Initially: OFF)

This set command will change any item command reference to phdate to mean phdate8, for assistance in converting to the newer phdate format found in PowerHouse version 8.19 and higher.

The set command:

```
>set date MapToPhdate8 on
```

only changes the reference to phdate8 in the item command, it does not change references that already exist in self-describing files nor does it change the data.

With this setting turned on any Item command reference, such as:

```
>item mydate,date,phdate
```

will actually mean phdate8.

DecimalSymbol

SET DecimalSymbol " "

(Default: period)

This setting is used to change the Decimal character which is used to edit the data in a field used by the $Number function.

```
   >set DecimalSymbol "."
```

The default character for the Decimal Symbol is a period.

Defer

SET DEFER ON | OFF

(Initially: OFF)

DEFER requests Suprtool to use DBCONTROL for faster "deferred-output" when doing Puts in the output phase or Deletes in the input phase. In deferred output mode, IMAGE writes are not posted immediately to disc. If the system crashes while you are in deferred output mode (i.e., you have a system failure or a nonrecoverable power failure), you must RESTORE your database from tape. If Suprtool stops due to an error, or if you abort Suprtool (i.e., break/abort or :abortjob) while the database is opened with deferred output, you do not have to restore the database.

Set Defer On requires that the database be opened in mode-3 (exclusive). If you do not open the database in mode-3, Suprtool prints a warning and does the Put or Delete task anyway (without calling DBCONTROL). The database creator can also enable deferred output for all users of a database with DBUTIL (enable base for autodefer).

DumpOnError

SET DUMPONERROR ON | OFF

(Initially: ON)

With DUMPONERROR, Suprtool attempts to produce a formatted listing of records that cause an IMAGE error. The information printed may include the input record

number, the output record number and the data values of the record. Suprtool uses current options of the List command to print the data values. If no List command is specified, Suprtool uses the List defaults.

EditSignNeutral

SET EDITSIGNNEUTRAL ON | OFF

(Initially: OFF)

There are three states for a sign for zoned-decimal (display) and packed fields. The states are positive, negative and neutral. This setting tells the $EDIT function to force a neutral field as positive such that the + sign will show up when an edit mask applied. In order to invoke this behaviour simply add the following set command to your task or if you want to invoke globally add it to your suprmgr file.

```
>set editsignneutral on
```

EditStopError

SET EDITSTOPERROR ON | OFF

(Initially: OFF)

An edit mask is limited to 32 characters in total for both numeric and byte type fields. If data overflows the edit-mask, by default Suprtool will fill that field with asterisks. To have Suprtool stop when it encounters an overflow when applying the edit-mask, just turn on editstoperror:

```
>set editstoperror on
```

This will force Suprtool to stop if there is data left over to place when applying the edit-mask. With numeric-type fields, leading zeroes do not cause overflow. With byte-type fields, spaces do not cause overflow.

Eofread

SET EOFREAD ON | OFF

(Initially: OFF)

Suprtool's Get command reads to the highwater mark for detail datasets, and to the capacity of the dataset for master datasets.

Set Eofread On forces Suprtool to read to the capacity for both master and detail datasets.

If Set Eofread is used, it must be specified before the Get command. Hint: Use HowMessy to see the highwater mark of your detail datasets.

FastRead

SET FASTREAD [ON | OFF]

This command has no effect on Suprtool for MPE.

Filecode

SET FILECODE [*number* **]**

(Initially: 0; Default: no change)

When Suprtool writes to an *old* file (Output,Append and Output,Erase are the only cases), it leaves the filecode of the output file unchanged. When Suprtool creates a *new* file with the Output,Ask or Output,Query option, it gives the file a special filecode (1071 and 1084 respectively). When the input file has a nonzero filecode, Suprtool carries that filecode over to the new output file. See "Output Command [O]" on page 233 for details on Output options such as Ask, Erase, and Query. To convert a self-describing file into a non-SD file, see "Suprtool and Self-Describing Files" on page 71.

In all other cases, *new* files created by Suprtool have a default filecode of zero. You can specify an alternate default filecode using Set Filecode.

Specifying Set Filecode forces Suprtool to create the output file with a specific filecode whenever it was going to use zero. To be effective, this Set command should be added to the Suprmgr file. To configure Suprtool to use the filecode 222 on output files, you would add this line to your Suprmgr file:

```
set filecode 222
```

This feature is useful for MPEX users who wish to manage Suprtool output files. For example, to find all of the Suprtool output files you would do the following:

```
:run mpex.pub.vesoft
%!listf @.@.@(code='222')
%exit
```

Of course, you can still use a :File command explicitly to override the filecode of the output file.

Filename

SET FILENAME Help | Link | Edit | Hint | Export | Outcount *filename*

You can change six of the seven built-in file names. The only filename you cannot change is Suprmgr.Pub.Sys. We recommend that you add these Set commands to Suprmgr.Pub.Sys. There are six options to Set Filename, one per built-in file name. SET FILENAME Help | Link | Edit | Hint | Export | Outcount

Examples

In these examples we assume that you want to move Suprtool and all its files to the Robelle group of the Dist account. We also assume that you wanted to rename the suprlink program from Suprlink to Linkprog.

```
>Set Filename Help suprhelp.robelle.dist
>Set Filename Link linkprog.robelle.dist
>Set Filename Edit dbedhelp.robelle.dist
>Set Filename Hint suprhint.robelle.dist
>Set Filename Export stexport.robelle.dist
```

Firstrec

SET FIRSTREC [0 | 1]

(Initial & Default: depends on file type)

FIRSTREC forces Suprtool to choose the first record number of the input file to be zero or one. If the input file is a KSAM file, this option overrides the setting in the KSAM key file.

This Set option is included primarily for RPG users. Any RPG program that takes advantage of KSAM record numbers assumes that the record numbers start at one and not zero, even though the KSAM file itself must be built with the first record defined as zero (RPG subtracts one before reading from the KSAM file).

Hints

SET HINTS ON | OFF

(Initially: ON)

By default, Suprtool provides all users with a "hint of the day" each time they enter Suprtool. These hints introduce people to Suprtool features that they may be missing. You must have the file Suprhint.Help.Robelle on your system. To disable hints, put Set Hints Off in Suprmgr.Pub.Sys.

Ifcheck

SET IFCHECK ON | OFF

(Initially: ON)

With Set Ifcheck On, the If command produces an error if any field used in the If command is not contained entirely within the input file record. For compatibility reasons, users may wish to disable this error checking by turning Set Ifcheck Off.

Ignore

SET IGNORE ON | OFF

(Initially: OFF)

IGNORE tells Suprtool to continue processing if there are errors in these situations: "duplicate-key" errors in writing to a KSAM file or IMAGE master dataset, "chain-head" errors in deleting entries from an input dataset or putting records to a detail dataset, a record changed before Suprtool could delete it, division by zero when doing an If, or invalid numeric data in an input file when doing an If.

InitExtents

SET INITEXTENTS ON | OFF

(Initially: OFF)

Set InitExtents setting is an experimental setting designed to reduce the length of the extent B-tree for the creation of output files. Turning Set InitExtents on will make the Initial Extents equal to the maximum number of extents on the creation of an output file. Keep in mind that that this will increase the size of the chunks that it may need for large files. Consequently, you may experience Out of Disc Space errors in some cases if large chunks of disc are not available.

ItemAbbreviateDate

SET ITEMABBREVIATEDATE ON | OFF

(Initially: ON)

The specification of the Date format within the Item command by default expects the entire keyword. For example, for the date format of Calendar, you would have to specify the entire token of Calendar. If you Set ItemAbbreviateDate On, you would only have to specify CAL for the Calendar date format.

ItemLock

SET ITEMLOCK *<fieldname>*

(Initially: Spaces)

Suprtool supports Item Level locking thru the Set ItemLock <fieldname> command. The set command must be specified after the Base and Get/Chain command has been specified and is only invoked for Delete and Update operations. For example:

```
!run suprtool.pub.robelle
base order,1,;
get dline
set itemlock item-num
del
out save,link
exit
```

This field/setting is reset after each task. Item-Level locking is not invoked for PUT operations.

There are some cases where Suprtool needs to use the "@" lock descriptor, when doing a dblock-mode 5 which is an unconditional Item Level lock but Dbutil show dbname locks will show it as a Dataset Level lock.

Interactive

SET INTERACTIVE ON | OFF

(Initially: depends)

If you run Suprtool from a session, Set Interactive is On. If you run Suprtool from a batch job or with Stdin or Stdlist re-directed, Set Interactive is Off. When it is On, Suprtool waits for answers to questions and continues processing even if there are errors. When it is Off, Suprtool aborts on any error, assumes the "correct" answer to any question, and generally acts as if there is not an intelligent being typing in the command. Suprtool chooses the "correct" answer, which allows the task to continue. In most cases, this is the default answer. However, there are cases where Suprtool picks a different answer from the default. For example, an "output filename,erase" command has a default answer of "no," but with Interactive Off, Suprtool uses the answer "yes."

However, if you run Suprtool on a Remote Session that was created from a batch job, Set Interactive is On even though you are NOT interactive. If you wish to have proper batch error processing, your first command after :Run Suprtool should be Set Interactive Off. Set Interactive Off is also useful when automating on-line tasks with usefiles:

```
:run suprtool.pub.robelle;parm=4; &
     info="set interactive off;use usefile"
```

LabelledTapeRewind

SET LABELLEDTAPEREWIND ON | OFF

(Initially: ON)

With certain versions of MPE/iX, the rewind operation (fcontrol 5) failed on SCSI DAT drives. Before Suprtool can read the data on a tape, it must read the labels first. That means the tape must be returned to the BOT. If the tape does not rewind properly, you can turn this off with Set Labelledtaperewind Off.

Limits

SET LIMITS [MPE ON|OFF] [READONLY ON|OFF] [Tablesize *size*]

[OVERRIDE ON|OFF]

(Initially NM: MPE ON, ReadOnly OFF, Tablesize 2,047 megabytes, Override OFF)

When Set Limits MPE is Off, you cannot execute **any** MPE command (e.g., :Purge). This is an irreversible option -- once disabled, it cannot be enabled again by the user. This option also applies to MPE commands executed inside Dbedit and Suprlink.

Table Size

The Table command allows you to load large tables. Once these tables reach the size of real memory on your machine, performance starts to degrade. Setting Tablesize restricts the total amount of table space to the specified number of megabytes. We suggest leaving at least one megabyte of real memory for MPE.

It is not necessary to enter both parameters in order to change one.

```
>set limits tablesize 2                    {MPE ability unchanged}
```

Due to internal restrictions in the CM version of Suprtool, the maximum Tablesize is fifteen megabytes. The NM version of Suprtool is restricted to 2,047 megabytes per table. We recommend that you set the Tablesize to be around 50% of your total memory.

Read Only

Suprtool normally allows any user with the proper access capabilities to add, update and delete records to a database. To prevent users from accidentally updating their database, we provide the following setting within Suprtool:

```
>set limits ReadOnly On
```

The ReadOnly setting, once turned on, cannot be turned off for the current run of Suprtool. This disables all commands that potentially change data for the specified database.

If Set Limits ReadOnly is enabled, the following commands from within Suprtool and Dbedit will not be processed and will return an error:

Suprtool

- Delete
- Put
- Update

Dbedit

- Add
- Change
- Delete

- Modify

You can Set Limits Readonly on the command line using the Info String and Parm option. For example the following command file can be used to restrict who has write access to a given database. In this example only the Mgr user is allowed write access:

```
IF HPUSER = "MGR" THEN
   run Suprtool.pub.robelle
else
   run suprtool.pub.robelle;parm=8;info='set limits readonly on'
endif
```

Override

Set Limits OverRide tells Suprtool for MPE to ignore any subsequent Set Limits TableSize commands. This was added since the new MPE table size is now 2 GB and works well without any limit. It was also added as Suprtool 4.4 and lower did not calculate the Set Limits Tablesize limit properly. So, if a user had Set Limits Tablesize 25, Suprtool would allow for a table greater than that old limit, while the new version of the table would stop with "Table Full" as prescribed by the command.

Overriding the TableSize, allows for customers to set in a global Suprmgr file and not have to search thru code to remove the specific command.

List

SET LIST *option value*

Use Set List to configure default values for the List command. You can configure the default date, time, and format for LaserJet listings.

List Date

SET LIST Date *number*

(Initially: 0)

When you select page headings with the List command by specifying a title, each page includes today's date. By default, this date is formatted as mmm dd, ccyy (e.g., Mar 20, 2000). Use Set List Date to specify a different default date format for future List commands (e.g., Set List Date 2). The valid date formats are as follows:

Value	Format	Example
0 (default)	mmm dd, ccyy	Mar 20, 2000
1	yy/mm/dd	00/03/20
2	mm/dd/yy	03/20/00
3	dd/mm/yy	20/03/00
4	dd mmmyy	20 Mar00

List PCL

SET LIST PCL [0|1|2|3|4|5|6]

(Initial & Default: 0)

Use Set List PCL to configure the default format for LaserJet listings. This option defines the List device as a PCL device and indicates the orientation and font for the report. Set List PCL affects only the List command; it is ignored by the Output command. PCL stands for Printer Command Language, which is an HP standard for printers. The LaserJet is one of the first PCL devices to be released by HP.

By default, Suprtool assumes that your List output device is not PCL-compatible (List PCL 0).

PCL 1. To print the Suprtool List output in Landscape mode (across the wide part of the paper) with the tiny Lineprinter font (16.66 pitch or 8 lines per inch), you should do the following (this setting prints 175 columns per line):

```
>:comment  Maximum of 175 columns with this font
>set list pcl 1
>:comment  You will need LaserJet with proper font cartridge
>list device laser123
```

PCL 2. To print the listing in Courier font, Landscape mode, 6 lines per inch, and 100 columns wide, use:

```
>set list pcl 2
>list device laserjet
```

PCL 3. This option selects the "standard" Portrait orientation with the Courier font of the LaserJet (80 columns across by 60 lines). You would use Set List PCL 3 when you insert a Font cartridge that overrides the default font (e.g., 92286F cartridge).

PCL 4. Selects Portrait orientation and Lineprinter font of the L cartridge (and others). This option prints 132 columns across the page by 80 lines.

PCL 5. Prints 80 columns on A4 paper by slightly narrowing the space between columns.

PCL 6. Prints tiny letters in Landscape mode on legal-size paper. This gives you 223 columns per line.

The PCL options, with the exception of PCL 5, were designed and tested with North American letter-size paper. Suprtool can adjust the number of rows and columns for each option to match A4 if you add 2000 to the PCL code. You can also select the ASCII character set (instead of the default Roman-8 character set) by adding 1000 to the PCL code. See "List Command [L]" on page 223 for more details.

Here is a complete table of the PCL codes:

PCL	L/P	Font	A4 paper Rows x Columns	Letter-size Rows x Columns	Notes
1	L	lp	58 x 188	60 x 175	
2	L	courier	43 x 110	45 x 100	
3	P	courier	64 x 77	60 x 80	"standard"
4	P	lp	85 x 128	80 x 132	
5	P	courier	64 x 80	60 x 80	A4-squeeze
6	L	lp	60 x 223	60 x 223	legal-size

L and P mean Landscape or Portrait orientation.

List Time

SET LIST Time *number*

(Initially: 1)

When you select page headings with the List command by specifying a title, each page includes the current time. By default, the time is in 24-hour format (e.g., 23:02). Use Set List Time to specify a different default time format for future List commands (e.g., Set List Time 2). The valid time formats are as follows:

Value	Format	Example
0	none	
1 (default)	24-hour	23:02
2	AM/PM	11:02PM

Lock

SET LOCK [*number* **]**

(Initially: 1; Default: 1000)

There are three different locking strategies that Suprtool uses for the Delete, Put, and Update commands. A complete discussion of locking is found in the IMAGE section of "Suprtool Issues and Solutions".

Set Lock 0. Locks the dataset(s) at the beginning of the task and unlocks it at the end. This option is good for middle-of-the-night tasks that are doing lots of Puts, Deletes, or Updates.

Set Lock 1 (initial value). Lock the dataset and unlock it around every Delete, Put and Update. This takes longer, but provides less database contention.

Set Lock *n*. Lock the dataset every n database transactions (Puts, Deletes, or Updates). This option is maintained as a compromise between Set Lock 0 and Set Lock 1.

MakeAbsent

SET MakeAbsent [On|Off]

Set MakeAbsent On, tells the prefetch module to make pages that are in memory to be released sooner.

```
>set makeabsent on
```

This feature is only relevant if prefetch is turned on with a setting of 1-5, which controls the size of a prefetch. Set MakeAbsent On has no impact if prefetch is turned off. For more information on this setting and its impact please read the following article from our web site at:

http://www.robelle.com/tips/st-performance.html

NLS

SET NLS [*number***]**

(Initially: 0 or NLDATALANG JCW)

Use Set NLS with MPE files to specify the language to be used for sorting Character-type fields (see "Native Language Support" on page 75). The *number* corresponds to an NLS language; you cannot use the NLS language name. Suprtool uses the value of the NLDATALANG JCW to set the default NLS language. The common language numbers are:

Number	Language
00	Native-3000
01	American
02	Canadian-French
03	Danish
04	Dutch
05	English
06	Finnish
07	French
08	German
09	Italian
10	Norwegian
11	Portuguese
12	Spanish
13	Swedish

NumBug

SET Numbug [On|Off]

The $number function had a bug whereby it would add on two zeroes and or bad data if the input number did not have a decimal point. We have fixed the bug so that the number function no longer adds the two digits on the end in error.

However, some users worked around this issue by doing the following:

```
>extract target = $truncate($number(conv-field) / 100)
```

Since some users, used this work around, the fix to the $number function will then return incorrect results. Therefore, we added the set numbug command to have Suprtool revert from the correct behaviour to continue to have the bug.

By default, Suprtool will just convert the number and not add on the data at the end, however, if you have used the work around then you can add the command:

```
>set numbug on
```

to the script directly or globally in your suprmgr file.

Openmode

SET OPENMODE [*number*]

(Initial & Default: 1)

By default, Suprtool opens databases in mode-1 when the mode is omitted from the Base command. Using Set Openmode, you can specify an alternate mode for the Base command. A common choice would be Set Openmode 5, so that Suprtool would allow only read-access to the database (disabling the Delete command, for example).

Oracle Rows

SET ORACLE ROWS *number*

(Not supported in MPE)

This option is not available in the MPE version.

Oracle OpenFix

SET ORACLE OPENFIX ON|OFF

(Initially: OFF)

This option is not available in the MPE version.

Oracle Integer

SET ORACLE INTEGER ON|OFF

(Initially: OFF)

This option is not available in the MPE version.

Oracle PassShift

SET ORACLE PassShift ON|OFF

(Initially: OFF)

This option is not available in the MPE version.

Oracle ZeroNull

SET ORACLE ZERONULL ON|OFF

(Initially: OFF)

This option is not available in the MPE version.

Pattern

SET PATTERN NEW | OLD

(Initially: NEW)

Prior to Suprtool version 3.1, there was no method of checking for the "@", "#", "?", or "~" characters in a pattern. Version 3.1 introduced a new pattern-matching routine, adding an escape character "&", and two new reserved characters "^" and "!". Old Suprtool tasks that look for the specific characters &, ^, or ! will not work with the new pattern-matching routine. Users who are concerned about this can add the following command to their Suprmgr.Pub.Sys file:

```
set pattern old
```

Prefetch

SET PREFETCH [*number* **]**

(Initially: 0; Default: no change)

Suprtool can read data directly from disc into memory with multirecord NOBUF reads. However, Suprtool is often slowed down on MPE/iX while waiting for the file system to satisfy its read requests. Suprtool can increase its throughput by using prefetch on MPE/iX and instructing MPE/iX's memory manager to read the data from disc to memory ahead of time. This way, Suprtool doesn't have to wait when it needs the data because the data it is already in memory.

The Set Prefetch command tells the memory manager how far ahead of Suprtool it should fetch data. Setting this number too low won't give the memory manager sufficient time to get the data into memory before Suprtool needs it. Setting the number too high may mean that on a busy system the data will be overwritten before Suprtool gets a chance to use it.

Due to changes in MPE memory management in more recent versions of MPE, performance improvements from prefetching data has decreased and in some cases, such as systems with a large amount of memory, prefetching shows little or no performance gain and may impact other users. It is for this reason that we have changed the default value to zero, or off.

In short our advice at this stage is for you to try different values of prefetch and see if you see performance improvements with some key tasks, however since we are not seeing the performance gains we did 15 years ago, when prefetch was implemented we are turning it off.

Suprtool/iX ignores the Set Prefetch value when reading from a remote file or database, or from a KSAM/V, KSAM/XL, circular, or RIO file. Set Prefetch is ignored by Suprtool/V.

Privmode

SET PRIVMODE ON | OFF

(Initially: ON)

Suprtool requires privileged mode to quickly scan IMAGE datasets. You can force Suprtool to read a dataset using DBGET by turning Set Privmode Off, but Suprtool runs about five times slower. You must specify Set Privmode Off before using the Base command. To enable fast serial reading, turn Set Privmode On and reopen the database with the Base command. This feature is intended for cases where bugs in MPE are causing Suprtool to fail.

Progress

SET PROGRESS Percent [*number* **] Minimum [** *number* **]**

(Initial & Defaults: Percent 5, Minimum 10000 (MPE/V) 50000 (MPE/iX))

The Set Progress command is used to turn the Suprtool progress report feature on or off. The PERCENT value specified tells Suprtool by which percentage increment to report the progress messages of any given input or output phase. The allowed range for set progress is from 0 to 25, the default is every 5 percent. If the PERCENT parameter is not specified, then the next parameter is considered to be the PERCENT

value. This is to remain compatible with some earlier pre-release versions of Suprtool.

The MINIMUM value is the minimum number of records that an input file must have in order for the progress reports to be printed out. If the MINIMUM value is set to 15000, then the input file must have at least 15000 records or else progress messages are not printed out for the entire task. This value allows the user to turn off progress messages when reading smaller files. The default value for the Suprtool/V version of Suprtool is 10000 records, and the default value for the Suprtool/iX version of Suprtool is 50000 records. To always print progress messages, just set the minimum value to 0.

Suprtool does not produce any progress messages under the following conditions:

1. Set Progress is zero.
2. Output is to $stdlist via the Output * or List commands.
3. Chain command has been specified.
4. The number of records from the input source is less than the minimum value.

Suprtool checks whether or not to print a progress message at the end of each buffer (for NOBUF/MR reads). Consequently, not all progress increments are reported for small files or datasets.

Suprtool reports the phase that it is in: whether input phase, sort phase, output phase or combined input/output phase (not sort).

The content of the progress messages is as follows:

- Percentage complete
- Phase and the total number of records processed
- Delta-Sec(Min) - the time elapsed from the previous message
- Wall-Sec(Min) - the total elapsed time
- CPU-Sec - the total CPU-Seconds at this point

When using the record selection feature of the Get and Input commands, Suprtool cannot be absolutely certain of the total number of records. Therefore, the percentage calculation is estimated.

Prompt

SET PROMPT *character*

(Initially: >)

PROMPT tells Suprtool which character to use for prompting. Any special character can be used as the prompt character. For example:

```
:run suprtool.pub.robelle
>set prompt %
%base store,5
```

RealMap

SETREALMAP ON|OFF

(Initially: ON)

This option has no effect in Suprtool for MPE.

Recover

SET RECOVER ON | OFF

(Initially: OFF)

This option can be used with an If command, which selects only records with a -1 in the first word, to extract deleted records from a KSAM file. This is not supported for KSAM/XL files prior to MPE/iX 4.0. On MPE/iX 4.0 and later, Suprtool can recover deleted records if you do not build your KSAM/XL files with the reuse option. If you use the reuse option, Suprtool can only recover deleted records where the space has not yet been reused.

Redo

SET REDO *filename*

(Initially: none)

Commands that you enter at the Suprtool prompt are saved in something called the redo stack. You can recall commands from the redo stack using other commands such as Before, Do and Redo. By default, the redo stack is stored in a temporary file but is discarded as soon as you exit. This means that your redo stack is not preserved accross invocations of Suprtool.

Set Redo allows you to assign a permanent file as the redo stack. The redo stack is then available for future Suprtool invocations. To assign MYREDO as a persistent redo stack, enter:

```
>set redo myredo
```

If the file does not exist, Suprtool creates it. If it already exists, Suprtool uses it. All the commands you enter from this point are written to the persistent redo stack. The setting is valid for the duration of the Suprtool session. As soon as you exit Suprtool, the setting is discarded. Next time you run Suprtool, you will get the temporary stack.

If the filename is not qualified, the redo stack is created in the logon group and account. This may be desirable if you wish to have separate stacks. If you wish to always use the same persistent stacks, you should qualify the name.

The Verify command shows which stack is currently in use. If it shows `<temporary>`, it means Suprtool is using the default stack. Anything else is the name of the file used on the Set Redo command.

Concurrency

When Suprtool uses the default temporary stack, it is only accessible to that particular instance of Suprtool. You can run as many Suprtool instances as you need and each one gets its own redo stack. With temporary stacks, you will never get into concurrency problems.

If you start using a persistent redo stack, you might start running into these type of problems. A persistent redo stack can only be used by one Qedit instance at any one point in time. If you try to use a persistent redo stack that is already in use, you will get:

```
>set redo myredo
EXCLUSIVE VIOLATION: FILE BEING ACCESSED  (FSERR 90)
Unable to open file for REDO stack
```

In this situation, Suprtool continues to use the redo stack active at the time so you can continue to work as normal.

Qedit also has the ability to have permanent redo stacks. It is advisable to have separate redo stacks for each product, as they will write commands to each others redo stack if you supply the same filename.

For example, if you use the command:

```
>set redo myredo
```

you will have a redo stack called myredo for your Suprtool commands. If you exit Suprtool and run Qedit and supply the same command set redo command your Qedit commands will be written to the same file that was used for your Suprtool commands.

By default, your permanent Redo stack will be built to 1000 commands. If you require a larger redo stack you can specify the size of the redo stack using a file equation prior to the set redo command.

```
>:file myredo;disc=10000
>set redo myredo
```

This will only be in effect the first time that the redo stack is built, subsequent file commands will not affect the size of the redo file. You can make the file bigger using tools such as MPEX's %altfile command, or by building a larger file and copying your current redo stack to the file.

Sortfast

SET SORTFAST ON | OFF

(Initially: depends)

Suprtool/iX has its own internal sort routines. These routines are as fast or faster than HP's sort routines, and do not require as much sort scratch space as HP's routines. Set Sortfast controls whether Suprtool's internal sort routines are called or whether HP's sort routines are used. In Suprtool/V, this option has no effect and defaults to Off. In Suprtool/iX, the Set Sortfast is On by default. In the unlikely event that problems are experienced with Suprtool's internal sort routines, HP's sort routines can be enabled with:

```
>set sortfast off
```

Squeeze

SET SQUEEZE [ON | OFF]

(Initial & Default: <neither>)

Turning Set Squeeze On forces Suprtool to always truncate the output file limit to the end-of-file when the output file is closed. Turning Set Squeeze Off forces Suprtool not to change the limit when closing the output file. By default or by specifying Set Squeeze (i.e., with no option) forces Suprtool to decide when to truncate the output limit to the end-of-file.

Suprtool will not squeeze a file limit if the input source is a file, the output file is not a new file, the output file is KSAM (CM, NM or KSAM64), or if the Ask or Num

query output options were specified. In all other cases, Suprtool squeezes the file limit down to the EOF.

Set Squeeze Off can be combined with the Numrecs command to automate job streams in which data is appended to the output file. For example,

```
>get d-sales
>numrecs 200%                          {output file will have extra room}
>output dsale
>set squeeze off                       {reserve room between EOF and LIMIT}
>xeq
>get d-history
>output dsale,append                   {Dsale has room for appending}
>xeq
```

Statistics

SET STATISTICS ON | OFF

(Initially: OFF)

STATISTICS causes Suprtool to print statistics at the end of each task. This can be useful for determining the effectiveness of Suprtool's Get versus Chain commands.

Subsystem

SET SUBSYSTEM ON | OFF

(Initially: OFF)

Normally, Suprtool ignores database subsystem access. Use Set Subsystem On to instruct Suprtool to examine the subsystem option of the database in the Base and Put commands. Subsystem access can be restricted to READ or NONE by using DBUTIL. If subsystem access is NONE, Suprtool does not open the database. If subsystem access is READ, the Suprtool Put and Delete commands are disallowed and the Dbedit Add, Delete, Modify, and Change commands are also disallowed.

Once you enable Set Subsystem On, you cannot disable it. This Set option applies to all databases; you cannot enable it for a specific database. Using the creator password overrides the subsystem flag and allows full access to the database. If you wish to take advantage of subsystem access, we recommend that Set Subsystem On be added to your Suprmgr file.

The following example restricts the STORE database to read access:

```
:run dbutil.pub.sys                    {must be DB creator}
>set store subsystems=read             {read access only}
>exit
:run suprtool.pub.robelle
>set subsystem on                      {check subsystem access}
>base store
Password [;]?                          {don't use creator}
>input infile                          {standard file input}
>put m-customer                        {Put will fail because we}
                                       {do not have write access}
Error:  You are not allowed access to this database
```

Suspend

SET SUSPEND ON | OFF

(Initially: varies)

SUSPEND instructs Suprtool to suspend after an Exit command.

The default is OFF when Suprtool is :RUN from MPE, when :RUN with Parm=32, or when invoked from the POSIX shell. Suspend is ON when Suprtool is :RUN from within another program (e.g., Qedit). In some cases, it is desirable to turn this option OFF to prevent several invocations of Suprtool from being present at the same time. Users who run Suprtool from within MENU systems or QUICK may want to turn Suspend Off, or :Run Suprtool with Parm=32.

ThousandSymbol

SET ThousandSymbol " "

(Default: comma)

This setting is used to change the Thousand character which is used to edit the data in a field used by the $Number function.

```
>set ThousandSymbol ","
```

The default character for the Thousand Symbol is a comma.

Userlabels

SET USERLABELS ON | OFF

(Initially: ON)

By default, Suprtool always copies user labels from the input file to the output file. User Labels are ignored when no Extract command has been issued and Set Userlabels is Off. However, if an Extract command is issued and Set Userlabels is Off, Suprtool still writes the Userlabels. If List Labels is requested, Suprtool prints the contents of the user labels before listing the data. If Userlabels is Off, Suprtool does no processing of user labels. To convert a self-describing file into a non-SD file, see "Suprtool and Self-Describing Files" on page 71.

Varsub

SET VARSUB ON | OFF

(Initially: OFF)

This option tells Suprtool to resolve any CI variables in any command. The default value for VarSub is Off for backward compatibility.

VarsubCompat

SET VarsubCompat [ON | OFF]

(Initially:OFF)

This command has no effect on Suprtool for MPE.

VarsubDebug

SET VARSUBDEBUG ON | OFF

(Initially: OFF)

Set VarsubDebug on will print out the line after the variable substitution has occurred. This setting only works if Set Varsub is on and Set varsubdebug is on.

```
setvar outfile &
:"/GREEN/SUPRTEST/filename901234567890123456789012345678901234567890123
45678901"
:run suprtool.pub.robelle
SUPRTOOL/iX/Copyright Robelle Solutions Technology Inc. 1981-2007.
(Version 6.2 Internal)  TUE, OCT 30, 2007,  2:58 PM  Type H for he
>set varsub on
>set varsubdebug on
>in file1sd.suprtest
vd:in file1sd.suprtest
>output !outfile,link,temp
vd:output /GREEN/SUPRTEST/filename90123456789012345678901234567890123
vd:2345678901,link,temp
```

The output is formatted into 74 byte chunks and printed with a preceding "vd:" so the "substituted" line is clear.

Warnings

SET WARNINGS ON | OFF

(Initially: ON)

Suprtool normally prints warning messages out to $stdlist. You can turn off these messages when you are running from batch by issuing a Set Warnings off command. If you are simulating batch mode with the Set Interactive Off command, you must do the Set Warnings off after the Set Interactive Off.

The default for this setting is On.

Xltrim

SET Xltrim ON | OFF

(Initially: OFF)

Suprtool will now release the disc space between EOF and the FLIMIT if Set Xltrim is on and Set Squeeze is either not set or Off. This releases the disc space after the EOF but leaves the Flimit alone. Set Squeeze on will still close the file such that the Flimit will be the same as the EOF.

The default for this setting is Off.

Sort Command [SO]

Specifies the next sort key via an IMAGE field name, or a field in a self-describing file, or a Defined field. See "Key Command [K]" on page 220 for sort keys specified by explicit byte position. Up to 20 Sort and Key commands can be specified per extract task. The first key entered is the major sort key.

SORT *field* [*(subscript)*] [DESCENDING]

(Default: Ascending order)

Field

The *field* specified must be an IMAGE data item that is a field of the input source, or a defined field, or a field from a self-describing file.

Subscript

If the field is a compound item (e.g., 2X25), the first sub-item is the default if no *subscript* is specified. You can sort on any sub-item by specifying the *subscript*. For example,

```
>sort address(2)                    {sorts on 2nd sub-item}
```

Descending vs. Ascending Order

By default, sorts are done in ascending order. *Descending* specifies that the field is to be sorted in descending order.

Examples

The most common use of the Sort command is to specify a sort field of a database field. You may use the Key command to specify all sorts. We recommend that the Sort command be used wherever possible. If the structure of your database changes, your Suprtool tasks still work if sort fields are specified with the Sort command:

```
>base store,5
>get d-sales                        {input from an IMAGE dataset}
>sort cust-account                  {primary sort field}
>sort purch-date,desc               {newest transactions first}
>output dsales                      {write the sorted records to an MPE disc file}
>exit
```

In the next example we sort an MPE file. We create a field using the Define command. Rather than using the Key command, we use the Sort command to specify the sort field. If the MPE file changes, only the Define command must be changed:

```
>input dinvent                      {input is from an MPE file}
>define a,11,2,int                  {"A" is an integer that starts}
>output outfile                     {    at the 11th byte of Dinvent}
>sort a                             {sort the input records by the "A" field}
>exit
```

With the *=dataset* form of the Input command, we can have even greater flexibility to use the Sort command. We assume that the MPE file Mcust has the same structure as the m-customer dataset. We sort the file using the first of the two street addresses in m-customer:

```
>base store,5,READER                    {open for read access only}
>input mcust=m-customer                 {same format as m-customer}
>sort street-address                    {a repeated field, but only}
>exit                                   {   the first address is used}
```

Notes

The Verify command shows all of the current Sort command values and the Reset command cancels them. If you have not defined any sort fields before the Xeq or Exit command, Suprtool performs a copy only, no sort.

It is important to note that if the field being sorted is the result of a $function, then you may not be sorting on the value of the field after the function has transformed the field. For example, the following task may not give you the result you expect:

```
>base store,5
>get d-sales
>def cust-accountx,1,6,byte
>ext cust-accountx = $edit(cust-accout,"zzzz99"
>sort cust-accountx          {sorting on transformed field before it has value from function}
>dup none keys
>output dsales
>exit
```

In this instance you will not be sorting on cust-accountx as transformed by the $edit function, but rather the first six bytes of d-sales. Therefore, it is important to note when you are using extract to transform a field, you will not be sorting on that transformed value.

Therefore, the way to do the transformation would be to either divide into two tasks or if you can logically sort on the field before the transformation in order to achieve the result, so the task could be:

```
>base store,5
>get d-sales
>def cust-accountx,1,6,byte
>ext cust-accountx = $edit(cust-accout,"zzzz99"
>sort cust-account                      {Note sorting on source field}
>dup none keys
>output dsales
>exit
```

or if two tasks are necessary:

```
>base store,5
>get d-sales
>def cust-accountx,1,6,byte
>ext cust-accountx = $edit(cust-accout,"zzzz99"
>output tempsales
>xeq
>in tempsales
>sort cust-acctx
>dup none keys
>output dsales
>exit
```

Table Command [TA]

Builds a table of values for testing in the If or Chain commands. There can be up to ten different tables. The Table command does not work with real-type keys.

TABLE *tablename, itemname, table-keyword, table-values*

Tables are used by the $lookup function of the If command and as input to the Chain command. The table keywords are Item, File, and Sorted. The total amount of table space is restricted by Set Limits Tablesize. Use the following scheme to select input records based on many data values:

1. Load a table with the values you are interested in.
2. Use the $lookup function of the If command to search the table.
3. Or, use the table with the Chain command to read a selected set of key values.

Adding Individual Values to a Table

To add a value for an item to a table, use:

TABLE *tablename*, *itemname*, ITEM, *value* [,*value*]

When you start entering the values for a table, you must enter all the values for that table before starting another table. Once you switch to another table, the previous table is "closed" and you cannot enter anymore values into it.

Parameters

tablename

Any identifier name up to sixteen characters long. This name can be the same as the name of a Define field or database itemname, but we recommend that you choose a unique name.

itemname

An item from the database specified in the Base command or a Define field. This cannot be a real-type item.

value

A specific *value* that must match the type of the *item*. String values are extended with blanks to the length of the item.

Examples

Suppose that you wanted to look for several part numbers. You could use the following If command:

```
>if part = "12345","67890","39201","92308","14892"
```

You could also load a table with the part numbers:

```
>table part-table,part,item,"12345"
>table part-table,part,item,"67890"
>table part-table,part,item,"39201"
>table part-table,part,item,"92308"
>table part-table,part,item,"14892"
```

and use a different If command:

```
>if $lookup(part-table,part)
```

Sometimes you need to look for all records that do **not** have any of a set of values,

```
>if part <> "12345","67890","39201","92308","14892"
```

You would use the same Table commands, but a slightly different If command,

```
>if not $lookup(part-table,part)
```

Values with Decimal Places

If the *itemname* for the table has implied decimal places, the Table command accepts decimal points and scales input values. For example,

```
>item cost,decimal,2                    {two implied decimal points}
>table cost-table,cost,item,10,10.5,10.75
>if $lookup(cost-table,cost)
>out out3                               {select records for 1000, 1050, and 1075}
>xeq
```

Adding Values from a File

You may need to look for hundreds of part numbers. The Table command accepts the table values from a file. The file must contain lookup values in exactly the same format as the itemname which describes the data. Duplicates are eliminated as they are loaded into the table. For a table consisting of values from a file use:

> TABLE *tablename*, *itemname*, FILE | SORTED, *filename*
> [,HOLD][,DATA(field1,field2,field3…)]

Parameters

itemname

The item determines the data-type and length of the key values in the table. By default, you can only load a table from a file when the item is the first field in the file. If you load a Table from a self-describing file, Suprtool first checks for the field in the self-describing file.

FILE vs. SORTED

The File option assumes that the file of table values is not sorted. Sorting a large file of values is slow. If the file is already sorted, use the Sorted option: Suprtool checks the records to make sure they are in ascending order.

filename

A valid MPE file name.

Hold

By default, the Xeq command resets all tables. Use the Hold option when using the same table in more than one extract task. When Hold is specified, the Xeq command does not reset the table. Hold applies to individual tables, not all tables.

```
>table part-table,part,file,partin,hold
```

Data

The Table command allows for data to be loaded along with matching key values.

```
     >table partab,part,file,partin,data(cost,desc)
```

You can specify up to 20 data fields as long as the total size of the key fields and data does not exceed 256 bytes. The Table file must be Self-Describing to use the data option.

When loading data into a table, Suprtool will eliminate the duplicate entries based on the key value, so some data values may not be loaded into the table.

For information on how to retrieve data from the table, please see the Extract command, or the Examples section of the Table command.

Examples

If all of your part numbers are in the file Partin, you use:

```
>table part-table,part,file,partin
>if $lookup(part-table,part)
```

The following example uses Suprtool to create a file of customer numbers that are in arrears. This file is then used to select the orders for these customers from the d-sales dataset. Instead of serially reading the file, we use the Chain command to take advantage of existing IMAGE search fields. The `cust-account` item occurs in both datasets. When the table is loaded, Suprtool obtains the `cust-account` field information from the self-describing custacct file.

```
>base store,5
>get  m-customer
>if   cust-status="30"                    {customers in arrears}
>out  custacct,link                       {self-describing output for the Table command}
>xeq
>table cust-table,cust-account,file,custacct
>chain d-sales,cust-account=cust-table
>output dsales
>xeq
```

The next example sorts selected records from the d-inventory dataset. Because the output file is not a self-describing file, we must use the Extract command to make the key-field the first field in the file. We use this file to load a table and select the records from the m-product dataset.

```
>get  d-inventory
>if   on-hand-qty<10                      {select records}
>ext  product-no                          {product-no must be first}
>ext  on-hand-qty                         {other fields follow}
>ext  last-ship-date
>sort product-no                          {sort by key value}
>out  invent                              {later, use this file in the Table command}
>xeq
>get  m-product                           {contains product description}
>table product-table,product-no,sorted,invent
>if $lookup(product-table,product-no)
>sort product-no
>output mproduct
>xeq
```

The next example shows how to select part numbers from flat file parts, using the file Partin. We use the Define command to define where the part-no field is in the record. This example assumes that the Part numbers are in the first six bytes of each file.

```
>input parts
>def part-no,1,6                          {define part-no}
>table part-table,part,file,partin        {product-no must be first}
>if $lookup(part-table,part-no)           {other fields follow}
>out  somepart                            {output to a file}
>xeq
```

Suprtool can load up to ten tables, either from separate files or the same file. The following example assumes that the files are self-describing.

```
>input customer
>table cust-table,custno,file,custfile
>table zip-table,zipcode,file,custfile
>if $lookup(cust-table,custno) and $lookup(zip-table,zipcode)
>output newcust,link
>xeq
```

Keep in mind that using multiple tables may be more memory intensive and require more resources.

Data Example

Your boss comes to you with a list of new prices for certain parts and asks you to update the Part-Master dataset.

Just load the new prices into a Table, index by the product number (prodno),then Extract the price field from each record and replace it with a $lookup on the table. Here is the code:

```
 >table newprices,prodno,file,bosslist,data(price)
 >get part-master
 >if $lookup(newprices,prodno)
 >update
 >extract price = $lookup(newprices,prodno,price)
  >xeq
```

We do the If $lookup to select only the parts which have new prices, then do Extract with $lookup to replace the existing price with a new one. The Update command forces a database update on each selected record and must come before the Extract command.

If you did not do the If $lookup then the price field would contain zeroes for those fields that did not have a matching record.

Notes on CM / MPE/V

The Xeq command clears any tables that are not held. Suprtool uses extra data segments to store the table values. You may not be able to load huge tables because of a lack of system resources (e.g., DST entries). To reduce the number of extra data segments Suprtool needs, you should configure the maximum extra data segment size to 32,764 words, using SYSDUMP on MPE V or SYSGEN on MPE/iX.

Searching huge tables is CPU-intensive. If you do this on-line, it seriously affects other users. While it is possible to load tables as large as 15 megabytes, we suggest that you make sure that there is enough real memory to hold the table. Remember that MPE takes at least one megabyte of real memory on Classic HP e3000s. Set Limits Tablesize restricts the maximum size of tables, so you can limit the total amount of table space to a specified number of megabytes.

In order to accomodate some large tables you may need to increase the value of Maximum number of XDS per process. The maximum value of this setting is 255 and can be changed using SYSDUMP on MPE V or SYSGEN on MPE/iX.

Notes on NM / MPE/iX

The Xeq command clears any tables that are not held. Searching huge tables is CPU-intensive. If you do this on-line, it seriously affects other users. While it is possible to load a table as large as 2047 megabytes, we suggest that you make sure that there is enough real memory to hold the table.

The Table command rejects RIO, KSAM, and CIR files when using the Sorted option.We do this because we cannot guareentee that we will get the records back in the correct order for anything other than standard files.The error returned when trying to use the Sorted option with one ofthese files is shown below:

```
Error:  Unable to open the table file
SUPRFILEINIT Error 7
File is KSAM, variable-length, or undefined-length
```

TRanslate Command [TR]

Translate command allows you to specify a From and To-character in decimal notation, which then can be subsequently used in the $translate function.

Translate “^from:^to” TOUNREAD TOREAD

The Translate command also has two special keywords to fill the translation table to help scramble data in byte-fields from readable to unreadable and back again to readable.

```
>in nametest;list;xeq
>IN NAMETEST.NEIL.GREEN (0) >OUT $NULL (0)
 NAME            = Neil Patrick Armstrong
IN=1, OUT=1.  CPU-Sec=1.  Wall-Sec=1.

>in nametest
>translate tounread
>ext name=$translate(name)
>list;xeq

>IN NAMETEST.NEIL.GREEN (0) >OUT $NULL (0)
 NAME            = Mtxo Kpecxrz Ocndeclmv
IN=1, OUT=1.  CPU-Sec=1.  Wall-Sec=1.
```

Total Command [T]

Totals specified fields in the selected input records. Totals can be written to $stdlist (default), a file, or to the List device.

TOTAL *field* [(*subscript*)] [*decimal-places*]

$file *filename* [APPEND | ERASE | TEMP]

$file $list

(Default: *subscript*=first sub-item, *decimal-places*=0)

Parameters

Each totaled *field* must be an IMAGE field name, or a field from a self-describing file, or a Defined field. Total specifies that a total value for the field be printed after processing the records. There does not have to be any output file (i.e., it can be $null) for a total to be printed. There can be up to 15 totaled fields.

The *subscript* is valid only for compound items. If no *subscript* is specified, the first field of a compound item is totaled. The *decimal-places* provides a decimal point when the final total is printed. If the Item command specifies the number of implied decimal places, the *decimal-places* parameter is not needed. The values within each field are assumed to be aligned.

$File Options

By default, totals are written to $stdlist. Use the $file option to have the totals written to a file or to the List device. When writing to a file, the default is to create a new file. If Suprtool cannot save the file, it produces an error. The Append, Erase, and Temp options are the same as in the Output command.

The file specified in $file must have a minimum record size of 48 bytes.

If you want the totals written to the end of the List device (the output of the List command), specify $list as the $file file name. All other options are ignored when $list is specified.

To write totals to the output file, use Total $file xxx,Append, where xxx is the name of your output file. You also need to turn Set Squeeze Off (otherwise, there will not be room for the totals). Appending totals to the output file only works if the output file is a disc file.

Examples

The first example prints the totals for a single field.

```
>in file1
>def total-field,1,5
>total total-field
>xeq
Totals (TUE, OCT 10, 2000,  4:30 PM):
TOTAL-FIELD                          611105+
```

The next example is identical to the previous one, except that we qualify the total with the number of decimal places.

```
>in file1
>total total-field,2
>xeq
Totals (TUE, OCT 10, 2000,  4:31 PM):
TOTAL-FIELD           6111.05
```

The previous example specified the number of decimal places by using the Total command. The next example is the preferred way to specify decimal places because it qualifies the `cust-bal` field with two decimal places for all Suprtool commands. This example also totals two fields (note that you need two Total commands) and shows how to total individual fields using subscripts.

```
>base  test
>get   dorder
>item  cust-bal,decimal,2
>total cust-bal(1)
>total cust-bal(2)
>xeq
Totals (TUE, OCT 10, 2000,  4:32 PM):
CUST-BAL(1)         143598.16
CUST-BAL(2)         155399.73
```

All the examples so far have shown the totals being written to $stdlist. The next example writes the totals to the file `totfile`.

```
>base   store
>get    d-sales
>total  sales-tax
>total  sales-total
>total  $file totfile
>xeq
```

To write the totals to the end of a Suprtool listing, use the $file option with $list as the file name. This example writes the totals to the end of the simple Suprtool report:

```
>base  store
>get   d-inventory
>list  standard
>total unit-cost
>total $file $list
>xeq
```

Sort Break Totals

Please refer to the Duplicate command on how to generate sort break totals.

Notes

Totals are formatted and displayed in a form that is understood by people. If you want to return total values to a program, see the *Calling Suprtool* user manual.

You cannot combine the Total command with the Total option of the Duplicate command.

The Total command prints out a date and time stamp on the title line. This is for audit purposes.

Update Command [UP]

Update one or more noncritical fields in an IMAGE dataset.

UPDATE [CIUPDATE]

Specify the fields to be updated and their new values with the Extract command. The Update command must be specified after the Chain or Get command, but before the Extract command(s). You can update part of an IMAGE field by defining the part you are interested in updating and then using the defined field name in the Extract command. The Update command can change values in critical fields (IMAGE search or sort fields). To update critical fields, you must specify the Ciupdate option, and CIUPDATE must be On or Allowed in the database.

Example

This example selects all inventory items with a unit-cost less than $10.00 and increases the unit-cost by five percent.

```
>get      d-inventory                  {input dataset}
>item     unit-cost,decimal,2          {implied decimal places}
>if       unit-cost < 10.00            {selection criteria}
>update                               {Update must come before Extract}
>extract unit-cost = &                {calculate the new …}
          unit-cost * 1.05            {… unit-cost value}
>xeq
```

This next example selects all inventory items with a part number of 12345677 and changes it to 12345678.

```
>get      d-inventory                  {input dataset}
>if       product-no = 12345677        {selection criteria}
>update ciupdate
>extract product-no = 12345678         {new unit-cost value}
>xeq
```

You can set CIUPDATE by using the DBUTIL Utility. To set CIUPDATE on just run Dbutil and use the Set command:

```
:run dbutil.pub.sys
>>set dbname ciupdate=on
>>exit
```

Notes

The only commands that can be combined with Update are selection commands (e.g., the If command). The following commands are not allowed: Delete, Duplicate, Key, List, Output, Put, Sort, or Total. You cannot update critical fields in master datasets. Update does not work with MPE or KSAM files.

If you are updating a packed or display field, remember that Suprtool uses unsigned values for non-negative numbers unless you add a leading plus sign to the number. See "Packed and Display Constants" on page 159 for more details.

Use Command [U]

Specifies a file of commands to be executed as a group.

USE[Q] *filename*

Database Date Items

A usefile makes your task easier by allowing common commands to be specified once in an external file. A common reason for usefiles is to isolate Define and Item commands for a database in one place. This makes future changes easier and prevents mistakes. In this example, we isolate all Item commands for dates from our database in a Suprtool usefile.

```
>use store.suprtool
define delivered,deliv_date
define purchased,purch_date
item delivered ,date ,yymmdd
item purchased ,date ,yymmdd
```

Dataset Field Definitions

In the store database, the street-address is a 2X25 item. Suppose that you always wanted to refer to the first and second part of the address with different names. The following usefile would access the m-customer dataset and define the two necessary fields:

```
>use mcust.suprtool
get m-customer
define street-address-1,street-address(1)
define street-address-2,street-address(2)
```

Info= and the Base Command

Our final example shows how to embed a database password in a usefile using the "?" password in the Base command.

```
>use store5.suprtool                {hint: "5" is the open-mode}
base store,5,?                      {"?" means read password from usefile}
Password?                           {reads password from usefile, but no echo}
```

Quiet Execution

By default, Suprtool displays the commands in a usefile as they are executed. The quiet option is not used in the examples above so that you could see the actual commands inside each usefile. Suprtool can execute commands *quietly* using the Useq command:

```
>useq store.use                     {no commands are listed}
```

You can also combine the Use command with Info=. In the following example, we specify the store database and execute the usefile with our Item commands:

```
:run suprtool.pub.robelle;parm=4; &
   info="base store.db.mis,1,?;useq store.suprtool.mis"
```

In this example we open the database in mode-1, but the user is prompted for the database password. In a job stream, the database password would be the first line after ":run suprtool". We then quietly execute all the Item commands for the store database (from the file Store.Suprtool.Mis).

Nested Usefiles

Usefiles may be nested. In other words, a usefile may use another usefile to a depth of ten files. For example, if the contents of the Usedef usefile has a references to Useext, both usefiles would be executed:

```
>in dsales
>use usedef
define delivered,deliv_date
define purchased,purch_date
item delivered ,date ,yymmdd
item purchased ,date ,yymmdd
use useext
ext cust-account
ext deliv-date
ext product-no
ext product-price
ext purch-date
ext sales-qty
ext sales-tax
ext sales-total
>xeq
```

Care must be taken when entering Use commands with a stacked command after a usefile reference. For example, if you enter

```
use usedef;def j,1,6,byte
```

the Define command will not be executed until *after* the Usedef and any other nested Use commands are finished.

Notes

The usefile may be an Editor /KEEP file,UNN or a Qedit workfile, but no more than 256 characters per record will be processed. For compatibility with Qedit, Useq can be abbreviated to UQ.

If a Use file ends with an ampersand, Suprtool will assume that you are continuing the current command on the next line, outside of the use file.

Userpause Command [USER]

The Userpause command prints a prompt message on the screen and waits for the user to press a key.

USERPAUSE [*"string"*]

(Default: read without a prompt)

Prints the *string* and waits for any key. Leading spaces in the string are printed.

Examples

In this example, we have a usefile that displays a message and then waits for the user before starting the task.

```
>q
>q "We will select all transactions over $10,000.  Since"
>q "there are many transactions, this task will take"
>q "some time (usually more than fifteen minutes)."
>q
>userpause "Press any key when you are ready to start."
```

Verify Command [V]

Displays the specifications that you have entered so far.

VERIFY [ALL | @ | VERSION | *command* [...]]

(Default: Input, Output, Sort, Numrecs, changed Set values)

Parameters

More than one command can be verified at once by entering several *command* names separated by a comma or a space. The format of the Verify output is organized into columns wherever possible.

For Verify All, Suprtool prints all of the information concerning the current invocation of Suprtool, including the value of the Set options and the Suprtool version number.

For Verify Version, Suprtool prints out the version information.

Verify with no parameters prints the current values for Chain, Get, Input, Output, and Key or Sort commands. It also prints those Set options which are not currently at their default setting.

Examples

```
>v
>verify
>verify input                    {current input file}
>verify if                      {selection criteria}
>verify all                     {all current options}
>verify version                 {version number of Suprtool}
```

Xeq Command [X]

Suspends entry of commands and begins the extract from the input source.

XEQ

Notes

After the Xeq, Suprtool processes the task and accepts commands to specify another task. Compare this with the Exit command, which stops Suprtool after processing the input. After an Xeq command, all parameters of Suprtool are reset to their initial values, except any database that is left open, the Set options, and the Defined fields even though their calculated offsets are not guaranteed to be correct for the next file processed.

Examples

```
>base cust
>get m-customer
>output mcstfile
>xeq                                    {copies m-customer to Mcstfile}
>get d-customer
>output dcstfile
>xeq                                    {copies d-customer to Dcstfile}
>exit                                   {terminate, no task to do}
                                        {last Xeq could have been Exit}
```

Calculator Command [=]

Evaluates an expression and prints the result in one of several formats.

=*expression* [,O | D | B | H | A | # | % | $]

Any command that begins with an equal sign (=) is treated as an *expression* to be evaluated. An expression consists of numbers and operators, followed by an optional display format.

The operators can be addition (+), subtraction (-), multiplication (*), division (/), or exponentiation (**). The value of the expression is printed immediately on $stdlist.

```
=20+15                                  {add two numbers together}
Result=35.0
=20*15                                  {multiply the same numbers}
Result=300.0
=20-15                                  {subtraction}
Result=5.0
=20/15                                  {divide, print precise result}
Result=1.33333333333
=20**15                                 {20 raised to the 15th power}
Result=.327680000000E+20
```

Order of Evaluation

Unlike most programming languages, the calculator always evaluates the calculation from left to right. This is similar to an electronic calculator, where each keystroke is operated on immediately. You can use parentheses to force the calculator to evaluate the expression in a different order.

```
=14+16+15/3                             {compute an average}
Result=15.0
=14+16+(15/3)                           {add 14, 16, and the result of 15/3}
Result=35.0
=14+((16+15)/3)                         {add 16 + 15, divide by 3, then add to 14}
Result=24.3333333333
```

Percentages

A number in the calculator *expression* may be followed by a percent sign (%). The calculator assumes that you want to qualify the number as a percentage.

```
=125*5%                                 {what is 5% of 125}
Result=6.25
=125+125*5%                             {add 5% of 125 to 125}
Result=12.5
=125+(125*5%)                           {oops, we needed to change the order}
Result=131.25                           {this looks like the answer we wanted}
```

The last two examples show the importance of the order in which calculator evaluates the expression. We needed to use parentheses to force calculator to evaluate our *expression* in the correct order.

Display Formats

A calculator *expression* may be followed by a comma and a display letter. The default is decimal (#) and the options are hex ($ or H), octal (% or O), double (D), ASCII (A) and binary (B). With these options, the result is treated as a 32-bit integer.

```
=10,o                                   {standard octal format}
Result=%000012
=-10,o                                  {negative number in octal}
Result=%37777777766
=100,h                                  {hexadecimal}
Result=$0064
```

In Double format, calculator prints the double result as two octal numbers (the way they appear in DEBUG). The first number represents the high-order 16-bits and the second number represents the low-order 16-bits.

```
=10,d                                   {treat result as two 16-bit octal words}
Result=%000000 %000012
=1000000000,d                           {high-order 16-bits are no-zero}
Result=%035632 %145000
=-10,d                                  {note negative value, 2's complement}
Result= %177777 %177766
```

In ASCII format, up to four characters are printed in Hex, Decimal, and ASCII display format.

```
=$2020,a
Result=$2020: 32,32 :"  "
=%20161 %72145,a
Result=$2071: 32,113:" q"  $7465:116,101:"te"
```

In binary format, the high-order 16-bits are examined. If these bits are not zero, they are printed as two groups of eight bits. A one (1) means that the bit is on and a zero (0) means that the bit is off. The low-order 16-bits are always printed as two groups of eight bits.

```
=10,b                                   {high-order 16-bits suppressed}
Result=%(2)00000000 00001010
=-10,b                                  {note negative value, 2's complement}
Result=%(2)11111111 11111111 %(2)11111111 11110110
=1000000000,b                           {high-order 16-bits are non-zero}
Result=%(2)00111011 10011010 %(2)11001010 00000000
```

Input Format

The calculator supports different input formats for numbers. Octal values are prefixed with a percent sign (%) and hex values with a dollar sign ($). An ASCII string of up to 4 characters is entered in quotes. The result of the last calculation is referred to using #.

```
=%12                                    {octal 12 or decimal 10}
Result=10.0
=%12,o                                  {octal input and octal display format}
Result=%000012
=$10
Result=16.0
=%177766                                {octal number that is really negative}
Result=-10.0
="abcd",h
Result=$61626364
=#,a                                    {use result of last calculation}
Result=$6162: 97,98 :"ab"  $6364: 99,100:"cd"
```

Programmers who make use of the MPE DEBUG software are often frustrated with the format that Double Integer numbers are printed. DEBUG prints them as two octal numbers. Calculator accepts two octal numbers as input and prints the result in standard decimal format.

```
=%35632 %145000                    {treat as one double integer value}
Result=1000000000.0
=%177777 %177766                   {negative double integer value}
Result=-10.0
```

Calculator Help

It may be difficult to remember all of the various options that the calculator offers. For this reason, you can obtain a short description of the calculator by entering the following:

```
=?                                 {? gives help}
                                   {prints a summary of = functions}
```

Suprtool Errors and Warnings

Two Types Of Messages

Suprtool prints two types of messages: errors and warnings. In both cases Suprtool is letting you know that it has encountered a condition of which you may want to be aware.

This appendix describes both kinds of messages and gives a partial list of warning messages.

Errors

Errors are defined as conditions which immediately prevent Suprtool from continuing, or which allow it to complete a task and then stop, because continuing would likely cause undesirable or erroneous results.

When Suprtool detects a serious error condition such as a syntax error in a command, a file system error, or a sort error, it prints an error message. For example,

```
Error:  Unknown command name, try HELP
Error:  Unable to open >OUTPUT file
```

Finding Errors Automatically

If you have software that scans spool files for error conditions, have it look for

```
"Error: ".
```

Batch vs. Interactive Mode

In batch mode, Suprtool aborts when an error message is printed, setting the JCW (job control word) to FATAL. This causes the rest of the job to flush (unless a !CONTINUE was in effect) and prevents subsequent job steps from using the results of the failed Suprtool run. In interactive mode, Suprtool prints the error message and prompts for the next command.

File System Errors

When a file system error occurs, Suprtool prints the MPE error message and asks if you want to see a full File Information Display. When Suprtool asks a question that requires a yes or no answer, "Y", "OUI", "JA", and "SI" are accepted as "YES", and any other answer is considered "NO".

If the output file is too small to hold all the selected input records, an error occurs and Suprtool terminates the extract task. In fact, any error during processing terminates the current task (exceptions: a "chain-head" delete error, a duplicate key value error, or bad data with an If command when Set Ignore is On). If Suprtool cannot Save the output file, it goes through a recovery procedure (see Output command).

Arithmetic Trap Abort

If Suprtool should Abort with Parm=99x, an error has been detected in the Arithmetic Trap Routine. This should never happen, so please report it to Robelle Solutions Technology Inc.

NUMRECS exceeded; some records not processed.

You specified a Numrecs and have reached it. This condition is considered an error if the input is from a source other than disc.

Command entered is not a valid Suprtool command.

and

MPE access has been disabled. See Set Limits command.

Normally, commands that are not valid Suprtool commands are passed off to the operating system. If access to the O/S has been disabled via the Set Limits command, these commands are no longer passed off. If the user does not precede the command with a colon, we assume that the invalid command was meant for Suprtool. If a colon precedes the command, we assume that the command was meant for the operating system. On HP-UX systems an exclamation mark can be used in place of a colon.

Output-ASCII not allowed with Duplicate None Keys

Not all processing options are allowed in all combinations. The ASCII option of the Output command, which reformats the output record, does not work with Duplicate None Keys. Dup None Keys assumes that the output record has the same data definitions as the input record.

Xxxx is not the search field of Yyyy

This message is issued by the Chain command when the search field specified (Xxxx) is not an index into the dataset (Yyyy). The field specified must be a TurboIMAGE search field or an MPE/iX third-party index.

Warnings

When Suprtool detects an unusual situation that it should bring to your attention, it prints a nonfatal warning message. For example,

```
Warning:  No input data specified
Warning:  DATABASE must be RESTORED if System Crashes
```

The following list explains the most common warnings.

New entries were added to the dataset.
Suprtool may have ignored some dataset entries.

or

The number of dataset entries was reduced.
Suprtool may have included some deleted dataset entries.

These two warnings let you know that while Suprtool was processing your task, other users added or deleted entries to the dataset.

Suprtool checks the dataset entry count before and after, and warns you if it has changed. Suprtool allows concurrent changes, but warns you when it happens. If you cannot accept multiple access, you must open the database exclusively (mode-4). If you consistently receive the warning about new dataset entries for a particular dataset, you should consider using Set Eofread (see the discussion under the Set command).

In many cases you can ignore these messages, but sometimes it indicates a serious problem with the current task. You may use Control-Y to stop the task from processing.

>OUTPUT has different record size.

The record size of the output file being written to has a different record size from the data being written by Suprtool. The size may be bigger or smaller.

This usually occurs when the file being written to is an existing file, or when a :File command is used to specify the record size explicitly. When Suprtool builds the output file for itself, it always uses exactly the record size needed to hold the output record.

If the output file size is smaller, your records are truncated. If the output file size is larger, your record is padded with blanks or binary zeros, depending one whether the file is ASCII or binary.

Output=Input task is not recoverable from this point.

This message is printed at the point that the input file has been re-initialized to accept output from the sort scratch file. If this task is aborted after this message prints, the data from the input file will be lost.

>OUTPUT file may be too small.

This message is printed whenever the number of input records (NUMRECS) exceeds the available number of output records. This often occurs when appending records to an existing file. If you have an If command, Suprtool does not know how many records will be selected. It assumes the worst case, that possibly ALL records will be selected.

Not all sort fields were extracted.
The sort information will not be written to the output Link file.

This warning occurs when you `>output filename,link` and are sorting by a field, but the field is not included in the list of extracted fields. Suprlink cannot use the file, but it may be a perfectly valid file for other applications.

NUMRECS exceeded; some records not processed.

You specified a Numrecs and have reached it.

Record selection in effect, percentage calculation is estimated.

You specified a Get or Input with record number selection and the percentage complete is estimated.

Welcome to STExport

Welcome to STExport

Welcome to STExport for MPE Version 6.2. STExport converts fields in a self-describing input file into an output file that can be imported into different applications.

Summary of the STExport commands:

Before	FLoating	Quote	Verif
CLean	Form	REDO	Xeq
Columns	HEAding	Reset	XML
DAte	HElp	Set	Zero
DElimiter	HTML	Sign	=expression
DO	Input	Spaces	:OS command
Escape	LISTREDO	SPaces	
Exit	Output	Use	

The minimum abbreviation of each command is shown in capital letters.

Installing STExport

STExport is installed as part of the Suprtool installation process. See the "Installing Suprtool" chapter of the *Suprtool User Manual* for more details on how to install both Suprtool and STExport.

Built-In File Names

STExport requires an external file for the Help command. STExport dynamically changes this file name, but you can use a :File command to override STExport's choice.

Default File Names

If you are running STExport on MPE V/E or MPE/iX, STExport finds the name of the STExport program file name (e.g., stexport.Pub.Robelle). STExport uses this name to determine the name of the other built-in file names. If STExport cannot call the procinfo intrinsic (e.g., this intrinsic doesn't exist on MPE V/R), it assumes you are running stexport.Pub.Robelle.

Account Name

The account name for all built-in files is the same as the one where STExport is running (e.g., if you :run stexport.Pub.Dev, the help file for STExport is stexport.Help.Dev).

Group Name

STExport examines the group name where STExport is running to determine the group name for the built-in file names. If the group name is Pub, STExport assumes help files are in the Help group. The same assumption is made for any group name where the first three letters are not Pub. When STExport is run from a group that starts with Pub (e.g., Pub), STExport assumes the help files have the same suffix (e.g., help).

Examples

Here are a few examples of the names that STExport would use for the STExport help file:

STExport Program File Name	STExport Help File Name
stexport.Pub.Robelle	stexport.Help.Robelle
stexport.Pub.Robelle	stexport.Help.Robelle
stexport.Pub.Account	stexport.Help.Account
stexport.PubRob.Account	stexport.HelpRob.Account
stexport.Robelle.Dist	stexport.Help.Dist

Changing Built-In File Names

You can use a :File command to tell STExport where the help file is located. Your :File command must use the file name that STExport dynamically assigned to the help file. For example, if STExport is called STExport.Robelle.Dist, you would use this :File command:

```
:file stexport.help.dist=stephelp.robelle.dist
```

Accessing STExport

How To Run STExport

To access STExport, type the following command:

```
:run stexport.pub.robelle
STExport/Copyright Robelle Solutions Technology Inc. 1996-2001
(Version 6.1)
$
```

After a short pause, STExport takes over your terminal and prints out some identifying information. You will notice that your command prompt has changed to "$", telling you that you have made it into STExport. STExport expects you to type command lines, ending each one with Return.

How to Xeq an STExport Task

Normally, you enter a series of commands. These commands specify the Input file, the Output file, and the formatting options. Finally, you enter an Xeq or an Exit command. This begins the actual STExport task.

If you entered the Exit command, STExport finishes the current task, then returns you to the operating system or the program that ran STExport.

```
$EXIT
End of program
:BYE
```

If you entered the Xeq command, STExport finishes the current task, then prompts you for another task. This continues until you enter the Exit command. If you wish to terminate STExport immediately (perhaps you are confused), enter Exit Abort. This terminates the STExport program immediately, without attempting any task.

Son Process

If you run STExport within Qedit or Select, you can retain the STExport process and quickly re-activate it later.

```
:run qedit.pub.robelle
/:r stexport.pub.robelle
$...
$exit
STEXPORT.PUB.ROBELLE is still alive. Okay to HOLD ? Y
/...
/:activate
STEXPORT.PUB.ROBELLE
$...
$exit
Program Held.  Use :ACTIVATE/:RUN;HOLD to re-run.
/...
```

Suprtool Export Command

You can access STExport from within the Suprtool program using Suprtool's Export command. Suprtool then runs STExport for you. All of STExport's commands are available to you in Suprtool. When passing STExport commands through Suprtool, you must preface each STExport command with *Export*. For example,

```
:run suprtool.pub.robelle
>base customer
>get m-customer
>sort cust-no
>output invoices,link
>xeq
>export input invoices
>export sign none                          {send "sign none" to STExport}
>export output invfile
>export exit                               {exit from STExport (not Suprtool)}
```

Refer to the Export command in the *Suprtool Commands* chapter of the Suprtool user manual for more information.

Preventing MPE Commands

If you want to prevent STExport from executing any MPE commands (e.g., :Purge), you Run STExport with Parm=8192. This feature is automatically invoked by Suprtool's Export command, when Set Limit MPE Off has been specified inside Suprtool.

```
:run stexport.pub.robelle;parm=8192
```

Exit with Verify

Some users find that they Exit from STExport inadvertently. For STExport to get user approval on Exit, Run STExport with Parm=64.

```
:run stexport.pub.robelle;parm=64
$e
Okay to exit [no]:
$
```

Preventing STExport from Suspending

If you run STExport from within HPDesk (and some other programs), STExport suspends on Exit but HPDesk does not notice. The next time you run STExport you will get a new copy of STExport. Eventually you will have many suspended copies

of STExport hanging from HPDesk, consuming system resources. Running STExport with Parm=32 forces STExport to terminate on Exit rather than suspend.

```
:run stexport.pub.robelle;parm=32
```

Job Control Word

STExport sets the system job control word (JCW) to a fatal state when STExport fails in a batch job. STExport sets only the high-order bit of the JCW. That is, it adds 32,768 to the existing JCW value. HP subsystems use the other bits of the JCW, so STExport does nothing to them.

Using STExport in Batch

STExport operates in session mode or batch mode. In batch usage, any "error" message causes STExport to quit, setting the Job Control Word to flush the remainder of the JOB. Warning messages do not cause an abort.

In batch mode, STExport does not prompt for missing information as it does in session mode. Instead, it attempts to choose the alternative that has the least chance of destroying valid data. For example, if the Output file is a duplicate file name in batch mode, STExport saves the new Output file with a "made up" name (OUTPUTnn, where nn is from 00 to 20), prints a warning message, and aborts.

Summary of Parm= Values

Value	Function
32	don't suspend, terminate completely
64	check with user before Exiting
8192	don't allow MPE commands

Values may be combined by adding them together. For example, Parm = 96 means "check with me before exiting, then when I do actually exit, terminate STExport completely instead of suspending".

STExportOutCount JCW

When STExport closes the output file, it sets a JCW named STExportOutCount with the number of output records. This is the same number reported in the total line (i.e., OUT=). You can use this JCW to control job stream execution by checking if the SprlinkOutCount JCW is non-zero. I there are more than 65,535 output records, STExport sets STExportOutCount to 65,535.

You can use the :Showjcw command to see the value of a JCW. For values greater than 16K, Showjcw displays either the word WARN, FATAL or SYSTEM followed by some digits. These words correspond to the following values:

```
WARN 16384

FATAL 32768

SYSTEM 49152
```

Add the value of the word to the number that appears with it for the true value of STExportOutCount. For example,

```
:showjcw STExportoutcount

STEXPORTOUTCOUNT = WARN8616

16384 + 8616 = 25000
```

The MPE/iX :Showvar command can also be used to see the value of a JCW. Showvar displays the full, correct number (e.g., 25,000) up to the maximum of 65,535.

STExport also sets two other JCWs: STExportOutCount1 and STExportOutCount2. These communicate the full OUT= value to the STExport2 interface.

STExportFullCount Variable

On MPE/iX, when STExport closes the output file, it also sets a variable namedSTExportFullCount with the number of output records. This is the same number reported in the total line (i.e., OUT=). The advantage of the variable is that if more than 65,535 records are written to the output file, the value of the STExportFullCount variable is not truncated.

Introduction to STExport

Importing Data

Use STExport to produce a formatted output file that can be used to import data into databases and applications. Other databases have different requirements for the format of input data. You will have to experiment with the various STExport formatting options to find a format that your particular database tool accepts.

Input File

STExport reads one input file and formats each input record into one record in the output file. The Input file must be a self-describing file (use the Output-Link option in Suprtool).

Dates and Decimal Places

Use Suprtool's Item command to specify date formats and the number of implied decimal places when you create the self-describing file. STExport uses this information to correctly format the information in the output file. See Appendix A for a complete example of how to use Suprtool's Item command and Output-Link option to create an input file for STExport.

Data-Types

Each STExport formatting command applies to all fields of a specific data-type (i.e., you cannot specify formatting field by field, only by type). For example, all numeric-type fields can be formatted the same.

The main data-types that STExport identifies are

Byte-Type:	STExport assumes that character information is stored in byte-type fields. By default, all byte-type fields are surrounded by quotes and trailing spaces are removed.
Numeric-Type:	The numeric data-types are integer, logical, floating-point, packed, and display. STExport converts the internal representation of each data-type into a string of ASCII digits. By default, all numeric-type fields have a leading sign and are variable length. Where appropriate, numeric-type fields are converted with a decimal point.
Floating-Type:	All commands that affect numeric-type fields also affect floating-type fields. In addition, you can use the Floating command to specify the format and decimal places for floating-type fields (i.e., Classic or IEEE floating-point numbers).
Date-Type:	If a field has a date format, STExport does extra formatting. By default, dates are formatted into yyyymmdd (e.g., 20001125).

Formatting Commands

Use the following table to determine which command applies to which data-type:

Command	Data-Type
Date	date-type
Floating	floating-type
Quote	byte-type
Sign	numeric-type
Spaces	byte-type
Zero	numeric-type

Commands

Many of STExport's commands, such as the formatting commands above, once set will retain their settings between tasks. Several other non-formatting commands will also retain their settings:

Command
Columns
DElimiter
HEAding
HTML

Each command and its options will remain in effect between any STExport task, unless specifically turned off.

For example, if a previous task has had custom Headings set with the Heading and Heading Add options, the Headings will remain in effect for each subsequent task until a new Heading option is entered.

Performance Considerations

On average, STExport is three-to-five times slower than Suprtool. This is the price we pay for having all of STExport's formatting features.

You can make STExport faster by doing the following:

1. Pre-select only the records you need with Suprtool. The fewer records STExport has to process, the faster it runs.
2. Use Suprtool's Extract command to select only the fields that you need to import in your final application. The fewer the number of fields in the input file, the faster STExport can format each record.

CI Variable Substitution

STExport is able to substitute any CI variables from any command line source, whether through interactive, use file or batch input.

In order to use this feature, issue the following Set command:

```
set varsub on
```

Variable Substitution is not on by default for backward compatibility.

Batch

Since the Streams facility (under default setups) will replace any "!" found in the first column of a job stream. Anytime you want to specify an entire line thru Variable Substitution you will need to leave a space before the variable is specified.

```
!setvar input "i dfile by message-no"
!run stexport.pub.robelle
$set varsub on
$ !input
```

Notes

For MPE commands some variables will be resolved twice when passed off to MPE, which will give different values for a variable.

Setting variables at the CI level:

```
MPEXL:setvar x 10
MPEXL:setvar y "!!x"
MPEXL:showvar [xy]
X = 10
Y = !x
```

Setting variables within STExport

```
$set varsub on
$setvar a 10
$setvar b "!!a"
$showvar [ab]
A = 10
B = 10
```

Because Stexport does one level of variable substitution prior to the command being passed off to MPE, setting variables in Stexport that involves other variables will give different results from MPE. We recommend that you set the variables prior to running Stexport, or that you temporarily turn off variable substitution with the Set Varsub Off command.

```
$set varsub off
$setvar a 10
$setvar b "!!a"
$showvar [ab]
A = 10
B = !a
$set varsub on
```

STExport Commands

General Notes

When you run STExport, it prompts for commands on $stdlist with a "$" character and reads command lines from $stdinx. STExport commands contain a command name followed by one or more parameters, and are patterned after the same commands in Suprtool.

In this chapter, we describe the STExport commands in alphabetical order. Following each command name in brackets is the minimal abbreviation for the command. For example: [I] for Input and [O] for Output.

Abbreviating

You may shorten the command to the first letter of the command name.

```
$v                                    {verify}
$x                                    {xeq}
```

Uppercase or Lowercase

You may enter the letters in either uppercase or lowercase, because STExport upshifts everything in the command line except literal strings within quotes ("abc") and file names. These two commands are identical:

```
$EXIT
$exit
```

Comments on Command Lines

Comments may appear at the end of any command line, when they are surrounded by braces. Many of the examples in this manual show comments at the end of each command line. You can enter a comment as the only item in a STExport command line. When continuing command lines, the comment can appear before or after the continuation character.

```
${ format reals with two decimal places. }
$input invoices
$floating fixed 2                       {Floating option}
$output invfile                         {produces the file we want}
$exit
```

STREAMX

STREAMX is a product from VESOFT that permits you to build flexible job streams. STREAMX contains a complete programming language with loops, prompts, and parameter substitution. A problem arises when trying to enter comments into a Suprtool batch job that will be submitted with STREAMX. Suprtool uses the {...} pair to delimit comments. STREAMX uses these same characters for expressions.

You cannot change Suprtool's comment character, but you can change the {...} characters in STREAMX. The following example shows how to change the STREAMX expression characters from {...} to ~...~:

```
!job example,user.acct
::setbraces ~~
!run stexport.pub.robelle
$input invoices                         {Input file to format}
$floating fixed 2                       {Option for floating-point numbers}
$output invdata                         {Produces the file we want}
$exit
!tell ~hpjobname~,~hpuser~.~hpaccount~;Example done.
!eoj
```

MPE Commands

STExport also accepts MPE commands, with or without a colon.

```
$:comment
$comment
```

For commands that are the same in both STExport and MPE, STExport executes the MPE command only if you type the colon. For example:

```
$help                                   {you get STExport help}
$:help                                  {you get MPE help}
```

MPE/iX Commands

STExport/iX executes any MPE command (e.g., Run), UDCs, and command files. **Caution:** programs that suspend, instead of terminating, are not killed by the HPCICOMMAND intrinsic.

File Names

STExport's Input and Output commands accept a file name in either MPE or POSIX format. File names starting with "/", "./", or "../" are treated as POSIX file names. All other file names are assumed to be MPE file names. MPE file names are upshifted and POSIX file names are not. POSIX file names are limited to a maximum of 240 characters. Here are some examples of POSIX file names:

```
:hello david,mgr.dev,david
:CHDIR SUBDIR
:run stexport.pub.robelle
$input ./file
$verify input
/DEV/DAVID/SUBDIR/file
```

Calculator

Any command line beginning with an equal sign (=) is treated as a calculator expression. This feature can be used to do other calculations without the need of an electronic calculator.

You can obtain a short description of the calculator by entering the following:

```
=?                                          {? gives a summary of = functions}
```

For a detailed description of the calculator and its options, see the Suprtool manual.

Control-Y

You can interrupt a STExport task with the Control-Y key (hold down Control while striking Y). STExport responds by telling you how far it has gotten (IN=, OUT=, etc.), and asking if you wish to stop. Hit the Return key to continue or type `YES` to stop the task.

Before Command [B]

Repeat any combination of the previous 1000 command lines, with or without editing.

BEFORE [*start* [/ *stop*]]
[*string*]
[ALL | @]

(Default: redo previous line)

(BQ=redo without change)

The Before command allows you to modify the commands before it executes them. If you don't need to change them, use BQ or Do.

The Before command uses Qedit-style control characters for modifying the commands. The default mode is to replace characters. To delete, use Control-D; to insert, use Control-B. If you prefer HP-style modify (D, R, I, and U), use the Redo command instead of Before.

Examples

```
$listf @.soruce                          {'source' is not spelled correctly}
NON-EXISTENT GROUP.  (CIERR 908)
$Before                                  {redo most recent command}
listf @.soruce                           {last command is printed}
         our                             {you enter changes to it}
listf @.source                           {the edited command is shown}
                                         {you press Return}

$listredo -10/
$before 5                                {redo 5th command in stack}
$bef 8/10                                {redo 8th through 10th}
$b listf                                 {redo last Listf command}
$b listf temp                            {redo "listf temp" command}
$b @temp                                 {redo last containing "temp"}
$before -2                               {redo command before previous}
$before -5/-2                            {redo by relative lines}
```

Modify Operators

If you wish to change any characters within the line, the modify operators are the regular Control Codes used in Qedit:

- Any printing characters replace the ones above.
- Control-D plus spaces deletes columns above.
- Control-B puts you into "insert before" mode.
- Control-A starts appending characters at the end of line.
- Control-A, Control-D, plus spaces, deletes from the end.
- Control-T ends Insert Mode, allowing movement to a new column.
- Control-G recovers the original line.
- Control-O specifies "overwrite" mode (needed for spaces).

Persistent Redo

Redo commands can be saved in a permanent file and can therefore be used from another session. You can use the **Set redo** command to specify a filename to save your redo commands. Please see the Set Redo command for details.

Clean Command [CL]

Specifies what characters to clean from a byte type field.

CLEAN [SPECIAL | <string> <range>]

(Default: None)

STExport will automatically clean all the byte type fields for a given SD file. To define what characters that need to be replaced you use the clean command with the character you want to clean in quotes. Since most of the characters that you will need to clean are unprintable, you can enter the decimal equivalent of the character. This is denoted by entering the "^" character in quotes preceding the decimal number of the character you wish to clean.

You can set the character with the command set cleanchar as shown below.

```
$ in mysdfile
$clean "^9","^10","^0","^7"
$set cleanchar " "
$out myexport
$xeq
```

Since the Cleanchar is by default set to space, the above task could simply be:

```
$in mysdfile
$clean "^9","^10","^0","^7"
$out myexport
$xeq
```

The SPECIAL keyword automatically defines Clean characters of Decimal 0 thru to Decimal 31.

```
$in mysdfile
$clean special
$out myexport
$exit
```

You can also specify a range of characters with the following syntax:

```
$in mysdfile
$clean "^0:^10"
$out myexport
$exit
```

The above task would clean all byte type fields of any characters from Decimal 0 (Null) to Decimal 10. (Line Feed)

Removing Bad Characters

You can have the Clean function clean the field, and instead of replacing with a space, STExport will essentially shift characters to the left by Setting the CleanChar in the following manner:

```
>Set Cleanchar "<null>"
```

STExport will pad the field that was cleaned with the appropriate amount of characters with a space at the end of the field.

Columns Command [C]

Specify whether fields are formatted into variable- or fixed-length columns.

COLUMNS FIXED | NONE

(Default: None)

Most PC software expects imported data to be in variable-length columns. Other database systems prefer data to be aligned in fixed columns. Use the Columns command to specify whether the output file has variable- or fixed-length columns.

Output File

The Columns command also affects the format of the Output file. If you specify Columns None, the output file will have variable-length records. If you specify Columns Fixed, the output file will have fixed-length records.

Date Command [DA]

Specify a specific date-format for all dates.

DATE NONE | *date-format* ["*separator*"]

INVALID ASTERISKS | NULL | "string"

(Default: yyyymmdd Invalid Asterisks)

Use the Date command to specify an output format for dates. Use the Invalid option to specify how invalid dates should be formatted in the output file. The advantage of the None option is that it formats all dates, whether they are valid or not. If you select a *date-format*, the default Invalid option replaces invalid dates with asterisks "*".

STExport must know which fields are dates and the format of each date. Use Suprtool's Item command and Output,Link option to specify the date information.

Date Format

The *date-format* can be one of:

- ccyymmdd
- yyyymmdd
- ddmmyyyy
- mmddyyyy
- yymmdd
- ddmmyy
- mmddyy
- aammdd

STExport converts each date field from its internal date format into the format that you specify.

Separator Character

By default, STExport formats all dates without a separator between the day, month, and year. Specify your own separator by enclosing it inside quotes after you specify the date format. The separator must be one character long. For example, to specify dates in ddmmyyyy format with a slash separator, use

```
$date ddmmyyyy "/"
```

To specify dates in yymmdd format with a dash separator, use

```
$date yymmdd "-"
```

Oracle Dates

Oracle dates contain both the date and the time. STExport formats the date, but not the time. If you specify Date None, Oracle dates will be treated as byte-type fields. Since Oracle dates actually contain binary data, the output is often unusable by other applications, unless you specify a specific *date-format*.

Invalid Dates

By default, all invalid dates are formatted as asterisks. STExport treats any date that does not have a valid century, year, month, or combination (e.g., February 29, 2000) as invalid. You can specify how you want STExport to format invalid dates by using the Invalid option of the Date command.

If you specify,

```
$date invalid null
```

STExport will produce a zero-length field if you specify Column Variable and spaces if you specify Column Fixed. If you want to specify an explicit string for all invalid dates, do so after the Invalid option. For example,

```
$date invalid "%%%%%"
```

will cause STExport to produce a string of five percent signs for any invalid date.

Example

First, use Suprtool to create the input file with the appropriate date attributes:

```
>get    d-sales
>item   deliv-date,date,mmddyyyy
>item   purch-date,date,mmddyyyy
>output dsales,link
>xeq
```

Then use STExport to read the dsales file. Specify Date ddmmyyyy "-" which causes all valid dates to be formatted in day-month-year format with a dash as the separator:

```
$input  dsales
$date   ddmmyyyy "-"
$output dexport
$xeq
```

Decimal Command [DEC]

Specify the format for the decimal place in numeric fields.

DECIMAL PERIOD | COMMA

(Default: Period)

The fields in the input file must have been created with decimal places, using Suprtool's Item command.

The Decimal command specifies what separator will be used to indicate the decimal place in numeric fields. In North America, the custom is to indicate the decimal place in numbers with a period (.). Outside North America, the custom is to indicate the decimal place with a comma (,). If the decimal place indicator is incorrect, it is harder to import files into other applications.

The Decimal command does not apply to floating-point fields.

Delimiter Command [DE]

Specify a delimiter, if any, that appears between each field in the output record.

DELIMITER NONE | COMMA | TAB | SPACE | *"string"*

(Default: Comma)

Use Delimiter Comma to create an output file in "comma-delimited" format (this is common for PC database applications). Use Delimiter Tab to tell STExport to insert the tab character between fields, instead of a comma.

If you have selected Columns Fixed, you will likely want to remove the delimiter by specifying Delimiter None. If you want some white space between fixed-length columns, specify Delimiter Space instead.

String Parameter

You can put anything inside quote characters to specify your own Delimiter. For example, Delimiter " , " would insert a space, a comma, and another space between each field in the output record. You can use either single- or double-quote characters to specify the delimiter (e.g., Delimiter " " and Delimiter ' ' are the same). The maximum length of the delimiter string is three characters.

Do Command [DO]

The Do command repeats (without changes) any of the previous 1000 commands.

DO [*start* [/ *stop*]]
[*string*]
[ALL | @]

(Default: repeat the previous command)

Commands are numbered sequentially from one as entered; the last 1000 of them are retained. Use the :Listredo command to display the previous commands. You can repeat a single command (`do 5`), a range of commands (`do 5/10`) or the most recent command whose name matches a string (`do list`). If you want to modify the commands before executing them, use Redo or Before.

Examples

```
$listredo
$do                                   {do previous command again}
$do 39                                {do command line 39 again}
$do 5/8                               {do command lines 5 to 8 again}
$do input                             {do most recent Input command}
$do -2                                {do command before previous}
$do -7/-5                             {do by relative line number}
$do 5/                                {do command lines 5 to "last"}
```

Notes

The Do command cannot be abbreviated.

Persistent Redo

Redo commands can be saved in a permanent file and can therefore be used from another session. You can use the **Set redo** command to specify a filename to save your redo commands. Please see the Set Redo command for details.

Escape Command [ES]

Specifies what character to escape the defined Delimiter, Quote or End of Line character.

ESCAPE DELIM | QUOTE | EOL |"string"|

(Default: None)

Many SQL importers allow you to add an escape character in front of characters that may mean something else to the import program. For example, if the import program thinks that the delimiter character is a comma, the importer may treat a comma in an address field as an indication to move to the next field, which will throw of the import.

Some import programs will treat the next character as data as opposed to a delimiter if the character is preceded by an escape character, such as a slash. Thus, when the field is analyzed by STExport the data that originally started as:

```
"Niagara Falls,Ontario, Canada"
```

would be transformed to be:

```
"Niagara Falls/,Ontario/, Canada"
```

This function will not work on fixed columns and can be invoked with the escape command:

```
escape delimeter quote eol "/"
```

The above command will take the defined delimeter, quote and Eol and escape with a "/", if found in any byte type field.

Excel Command [EXC]

Use the Excel to produce columns of data that when imported will preserve spaces or leading zeroes.

EXCEL PRESERVE <*fieldname*>

Example

STExport can generate columns that are imported into Excel in such a way that leading zeroes are preserved. While the format produced is not traditional CSV, the format will produce a field in the form:

```
="00055555"
```

This form when imported into Excel will preserve the leading zeroes. In order to invoke this format, the Excel command has very simple syntax:

```
$in filexcel
$col fixed
$quote double
$zero leading
$excel preserve newchar int-field
$out *
$xeq
```

These simple commands will generate a file that will have the usually formatted fields as well as some fields formatted specifically for preserving spaces and leading zeroes in Excel.

The result of such an STExport task will look as follows:

```
="    11111 ",=" 01111", 0000011111,+00000011111
="    11111 ",=" 02222", 0000022222,+00000022222
```

Exit Command [E]

Exit STExport in one of three ways. By default, perform the current task, if any, then leave STExport. Users are often frustrated when they exit STExport after specifying part of a task and STExport starts processing the task. To avoid this situation, use the Abort or Suspend options to exit STExport conveniently without executing the current task.

EXIT [ABORT | SUSPEND | XEQ]

(Default: Xeq)

Typing Exit with no parameters means Exit Xeq. STExport recognizes special command names which specify both the Exit command and an exit option (e.g., ES means Exit Suspend).

Exit Abort [EA]

Cancels the current operation and terminates STExport. The Exit command without parameters always attempts to perform the task currently specified, while Exit Abort cancels the task and terminates immediately. Should STExport be executed as a son process, Exit only suspends STExport, while Exit Abort actually terminates the process.

Examples

```
$:comment.  You began to specify an input file, stopped for
$:comment.  coffee, and decided to cancel the task
$:comment.  upon your return.
$input invoices
... coffee break ...
$exit abort                              {cancel the task and terminate}
End Of Program
```

Exit Suspend [ES]

When running STExport as a son process (e.g., from Qedit), it would be nice to suspend STExport without executing the current task. Exit Suspend does this. After returning to STExport, all specifications for the current task are still in effect.

Examples

```
:run qedit.pub.robelle
/:run stexport.pub.robelle
$input invoices
$floating fixed 2                        {start specifying options}
$exit suspend                            {return to Qedit without an Xeq}
STEXPORT.PUB.ROBELLE is still alive. Okay to HOLD ? Y
/...
/:activate STExport
STEXPORT.PUB.ROBELLE
$output invdata                          {continue specifications}
$xeq                                     {execute the current task}
```

Exit Xeq [EX]

To perform the current task, you can either use Xeq (which leaves you inside STExport, ready to define another task) or Exit Xeq (which leaves STExport when done with the task).

After the STExport task completes, STExport either terminates, or suspends and awakens a father process (i.e., :Run from within Qedit). Exit Xeq is the default option (i.e., specifying exit starts execution of the current task).

Examples

```
:run stexport.pub.robelle
$exit                                   {no input was specified}

:run stexport.pub.robelle
$input invoices
$floating fixed 2
$output invdata
$exit                                   {format and stop}
```

Excel Command [EXC]

Use the Excel to produce columns of data that when imported will preserve spaces or leading zeroes.

EXCEL [PRESERVE <*FIELDNAME*>]

STExport can generate columns that are imported into Excel in such a way that leading zeroes are preserved. While the format produced is not traditional CSV, the format will produce a field in the form:

```
="00055555"
```

This form when imported into Excel will preserve the leading zeroes. In order to invoke this format, the Excel command has very simple syntax:

```
$in filexcel.stexptst
$col fixed
$quote double
$zero leading
$excel preserve newchar int-field
$out *
$xeq
```

These simple commands will generate a file that will have the usually formatted fields as well as some fields formatted specifically for preserving spaces and leading zeroes in Excel.

The result of such an STExport task will look as follows:

```
="    11111 ",=" 01111", 0000011111,+00000011111
="    11111 ",=" 02222", 0000022222,+00000022222
```

Floating Command [FL]

Specify the format and the number of decimal-places for floating-point fields.

FLOATING DEFAULT |
FIXED *decimal-places* |
SCIENTIFIC *decimal-places*

(Default: Default)

By default, STExport formats floating-point fields (classic floating-point or IEEE floating-point) into either a fixed number or into scientific notation. Which notation STExport chooses, depends on the value of each field in each input record. You can force STExport to choose either scientific or fixed notation and the number of decimal places for all floating-point numbers. You cannot specify these options for a specific field or make them different for 32-bit versus 64-bit floating-point numbers.

Fixed Format

Use Floating Fixed to force all floating-point numbers to appear in a fixed format. You specify the maximum number of digits to the right of the decimal point. If you specify Floating Fixed, STExport does not remove trailing zeros from the formatted numbers. If you specify Columns Fixed, all floating-point values will be aligned along the decimal point.

Scientific Format

Use Floating Scientific to force all floating-point numbers to appear in scientific notation. You must specify the number of digits to the right of the decimal point. The Scientific option formats the number with all significant digits to the right of the decimal-point followed by the exponent (e.g., "0.47832E-10").

Notes

Both the Fixed and Scientific options attempt to round the number to the specified number of decimal-places within the maximum width for each floating-point data-type. If STExport cannot format a floating-point field in the specified number of decimal-places, the number appears as asterisks "****".

Form Command [F]

Display the fields in a self-describing file.

FORM [*filename*]

If no file name is specified, the fields in the input file are displayed. The display shows the field type and field length in IMAGE notation. An I1-field is a single integer. Packed-fields show the number of nibbles (subtract one to obtain the number of digits). Byte and zoned-decimal fields show the byte length.

When showing the form of a self-describing file, STExport shows the byte offset of each field after the subcount, type, and sublength. The first field always appears at offset one.

There are two types of self-describing files. One type is produced with Suprtool's Query output option. You produce the other type with the Link output option. The Form command shows the internal self-describing version number, enabling you to tell the difference.

A.00.00 - Query Output Option

Compound fields have a question mark for the type, and the length is the number of bytes in the field. Sort information about the file is missing. Here is an example form listing:

```
$form custfile
File: CUSTFILE.EXAMPLE.ROBELLE   (SD Version A.00.00)
   Entry:                Offset
      CHARACTER          X5    1         {length is five bytes}
      ZONED              Z5    6         {room for five digits}
      INTEGER            I1   11         {single integer}
      DOUBLE             I2   13         {double integer}
      PACKED             P6   17         {room for five digits}
      QUAD               I4   20         {eight-byte integer}
      REPEATINT          ?6   28         {compound field}
      LOGICAL            K1   34         {single logical}
      DBLLOG             K2   36         {double logical}
Limit: 10000  EOF: 15  Entry Length: 44  Blocking: 64
```

B.00.00 - Link Output Option

These self-describing files contain information about how the file was sorted. Compound fields are handled correctly, so the Form command shows compound fields just as you would see them in IMAGE. The Item command in Suprtool identifies the date format or the number of decimal places of an item. The Link output option saves the date and decimal attributes as part of the field description:

```
$form custfile
File: DATAFILE.EXAMPLE.ROBELLE   (SD Version B.00.00)
   Entry:                Offset
      CHARACTER          X5    1  <<Sort #1 >>
      REPEATINT         3I1    6        {compound field}
      DATE               J2   12  <<YYYYMMDD>>
      DOLLAR             P6   16  << .2 >>
Limit: 10000  EOF: 15  Entry Length: 16  Blocking: 64
```

Formout File

The Form command writes all output to the file Formout. This file defaults to $stdlist. You can redirect this file to a line printer or a disc drive. If you redirect the Formout file to a disc file, Suprtool assumes a temporary file by default.

```
>:file formout;dev=lp
>form custfile                          {writes to line printer}
>:file formout;dev=disc
>form invfile                          {writes to temporary file}
```

Heading Command [HEA]

Specify a heading, if any, that appears as the first record of the output file.

HEADING NONE | FIELDNAMES |
string | ADD *string* | COLUMN *string*

(Default: None)

When importing data into other applications the first line of the import file is often treated as field names or headings. Use the Heading command to specify what STExport should write as the first line of the output file.

Field Names

If you specify Heading Fieldnames, STExport creates a default heading. This heading is constructed by using the field name of each field in the input file. The Fieldname option uses the formatting options that apply to byte-type fields to determine the final format (e.g., the Quote command).

STExport produces multiple field names for compound fields. For compound fields, the repeat count is used to determine the number of field names. The repeat count is appended to the field name, starting with one, until all the field names have been generated.

User Specified Heading

You can specify your own heading line by doing:

```
>heading "your heading"
```

Because the maximum length of an STExport input line is 256 characters, you may not be able to specify a long heading with a single Heading command. Use Heading Add to add additional strings to your heading:

```
Heading     "Account"                    {Note no Add in first string}
Heading Add "First Name "
Heading Add "Last Name "
Heading Add "City "
Heading Add "State "
```

If you specify your own heading, STExport does not attempt to apply any formatting options. If you need each field in the heading line to be surrounded by quotes and separated by commas, you have to supply these yourself. For example,

```
Heading     '"Account"'                  {Note no Add in first string}
Heading Add ','
Heading Add '"First Name"'
Heading Add ','
Heading Add '"Last Name"'
Heading Add ','
Heading Add '"City"'
Heading Add ','
Heading Add '"State"'
```

Column Headings

It is difficult to get headings right when you have to specify all the quotes and delimiters with the Heading Add option. Instead, use Heading Column to specify individual column headings without having to type formatting information. STExport then uses the current quote and delimiter settings in the heading.

For example, if you specify:

```
Heading Column 'Account'
Heading Column 'First Name'
Heading Column 'Last Name'
Heading Column 'City'
Heading Column 'State'
```

and Quote Double and Delimiter Comma are in effect, then the heading STExport produces will be:

```
"Account","First Name","Last Name","City","State"
```

Notes

You cannot combine the Add and Column options. You must specify one or the other. If you start with Heading Add and then later specify Heading Column, STExport erases the heading you created with Heading Add and starts over with the first column that you specify with Heading Column. Similarly, if you start with Heading Column, a Heading *string* or Heading Add will start over with a new heading.

Help Command [H]

Show what commands and options are available in STExport.

HELP [*command* | *keyword* [,*section*]]

(Default: browse through the entire help file)

Command Help

If you specify any parameters, Help first assumes that you want help on a specific STExport command. If you know the structure of the help file, you can specify one of the keywords under the command name.

```
$help sign                          {help on the Sign command}
$help sign,trailing                 {trailing section of the Sign command}
```

Keyword Help

If we cannot find any help in the "Commands" section of the help file, we assume that you specified one of the outer-level keywords in the help file. To see this list of keywords, type help with no parameters. You see a short introduction to STExport and then a list of keywords. You can specify any of these keywords on the Help command. You can also specify a subkeyword.

```
$help intro,input                   {input section of Introduction}
```

Quick Help - HQ

HQ asks STExport to look under the keyword Quick in the help file. Quick contains the text from the STExport Quick Reference Guide, offering the experienced user a quick review of the syntax of any command.

```
$hq input                           {quick description of Input}
```

Notes

If no parameters are specified, Help allows you to browse through the help file, stexport.Help.Robelle. The Help command uses the Qhelp subsystem from the QLIB. For "help in help", type "?" when you see the Qhelp prompt character ("?"). The help file is organized into levels. To go back to the previous level, press Return. Press F8 to exit the Qhelp subsystem and return to STExport.

HTML Command [HT]

Use HTML to produce Web pages for either Internet or Intranet applications.

HTML NONE | PREFORMATTED | TABLE |
TITLE *"string"* |
HEADING *"string"*

(Default: None)

Web applications expect data in a special format called the Hypertext Markup Language (HTML). Use the HTML option to request that STExport format the input file into HTML format.

Example

```
$html table title "Product Listing"
```

Maximum Size of HTML Files

Web browsers often cannot process large documents. The maximum size depends on the browser, the version of that browser, the operating system it is working on, and how much physical memory is present on the client machine. We suggest that you limit your Web pages to less than 1,000 lines and restrict the number of columns, unless you are certain that your users can handle larger files. This advice reflects not so much a limitation of STExport, but a limitation of how Web browsers work.

Preformatted Format

To preserve the columns and spacing of each output line, use the HTML Preformatted option. This option puts an HTML <pre> tag around all the data in the input file. Most Web browsers will display preformatted text in a fixed-width font such as Courier. Therefore, if you specify HTML Preformatted, you should also select Columns Fixed.

Table Format

Use HTML Table to create output in HTML table format. STExport creates tables with a border between each column and row. Tables make it easier to read tabular information, but some older browsers do not support tables.

If you specify HTML Table, all byte-type fields are left-justified, and all other fields are right-justified. If you use Heading Column or Fieldnames, the column headings are specified with HTML table heading tags. Most browsers highlight the column headings in some way, such as bold text centered over the column.

Title

All HTML documents must have a title. By default, STExport uses the title "This is the Title". You should specify your own title using the Title option.

Heading

The heading appears before the column headings and data from your input file. By default, there is no heading. Use the Heading option to specify your own heading.

Column Headings

If you specify HTML Table, use the Heading command to specify column headings for HTML output. The Heading Fieldnames option will produce acceptable column headings, but it is better to use Heading Column to specify a string for each of the fields in your input file.

Roman-8 Characters

HP e3000 and HP 9000 computers use the Roman-8 character set. Web pages must use the ISO-8859-1 character set. The characters in the Roman-8 set are similar to, but not identical with, the ISO-8859-1 character set.

When formatting byte-type fields, STExport attempts to convert any Roman-8 input character into the corresponding ISO-8859-1 character. STExport also tries to make reasonable conversions such as:

Symbol	Converted To
>	&gt
<	&lt
&	&amp

Those characters that cannot be converted are dropped from the output. The following characters cannot be converted:

Symbol	Roman-8 value	name
`	169	grave mark
^	170	circumflex
~	172	tilde
f	190	function symbol
ß	222	beta symbol
Š	235	capital-S, Icelandic
š	236	small-S, Icelandic
Ÿ	238	capital-Y, umlaut

Notes

If you specify HTML Table, STExport sets:

- Quotes None
- Delimiters None

If you specify HTML Preformatted, STExport sets:

- Quotes None
- Delimiters Space
- Columns None

In either case, any changes cause STExport to print a warning to let you know that these options have changed. If you do want quotes around byte-type fields or delimiters between fields, specify those options after selecting the HTML option.

Input Command [I]

Specifies the primary input file.

INPUT *filename*

There can be only one Input file per task. The Input file must be a self-describing file, which should be created by Suprtool using the Output-Query or Output-Link option. If you want STExport to format date-fields and implied decimal places, you must use the Output-Link option of Suprtool when you create the file for input to STExport.

Every record in the input file is formatted into a corresponding record in the output file. It is best to have Suprtool Extract only the fields you actually need. Only those fields needed for import into the final application should be present in the Input file.

Json Command [J]

Specifies STExport to generate Json output. Use the JSON to produce Java Script Object Notation documents for either Internet or Intranet applications.

JSON

OBJECT "string"

ONEPERLINE

Example

STExport can generate JSON output with just a few commands.

```
$input file1sd
$JSON
$output myJSON
$xeq
```

These four simple commands will generate a file that can be read by various applications. The result of such an STExport task will look as follows:

[{"CHAR-FIELD":"11111","INT-FIELD":1111,"ZONED-FIELD":11111}]

Object

The Object option allows the JSON data to be wrapped in a specific Object description.

```
JSON Object "Json object"
```

Looks like this:

```
{"Json object":
[{\"CHAR-FIELD":\"11111",
\"INT-FIELD"\:1111,
\"DBL\-FIELD"\:11111,
\"PACKED\-FIELD"\:+1111,
\"PACKED\.-FIELD"\:+11111,
\"QUAD\-FIELD"\:11111,
\"ID\-FIELD"\:1,
\"LOGICAL\-FIELD"\:111,
\"DBLLOG\-FIELD"\:11111,
\"ZONED\-FIELD"\:11111
}]
}
```

Note that the example of the Output has one field per line with data. Normally this would have to be specified via the command line but the data is shown this way simply due to space constraints.

OnePerLine

For files that have many fields you may want to consider using the OneLine option of the JSON command:

```
JSON OnePerLine
```

STExport will put each field and data on one line with the appropriate beginning and end notation.

```
[{\"CHAR\-FIELD"\:"11111",
\"INT\-FIELD"\:1111,
\"DBL\-FIELD"\:11111,
\"PAC\KED\-FIELD"\:+11111,
\"PAC\KED\.-FIELD"\:+11111,
\"QUAD\-FIELD"\:11111,
\"ID\-FIELD"\:1,
\"LOG\ICAL\-FIELD"\:1111,
\"DBL\LOG\-FIELD"\:11111,
\"ZONED\-FIELD"\:11111
}]
```

Multiple Json Commands

You can enter multiple JSON commands per task to set the JSON options you require.

```
$in file1sd
$JSON Object "Json object"
$JSON OnePerLine
$out *
$xeq
```

An example of the output generated by the above commands is as follows:

```
{"Json object":
[{"CHAR\-FIELD"\:"11111",
\"INT\-FIELD"\:1111,
\"DBL\-FIELD"\:11111,
\"PACKED\-FIELD"\:+11111,
\"PACKED\.-FIELD"\:+11111,
\"QUAD\-FIELD"\:11111,
\"ID\-FIELD"\:1,
\"LOGICAL\-FIELD"\:1111,
\"DBLLOG\-FIELD"\:11111,
\"ZONED\-FIELD"\:11111
}]
}
```

Listredo Command [LISTREDO]

The Listredo command displays any of the previous 1000 commands.

LISTREDO [*start* [/ *stop*]] [;ABS] [;OUT=*file*]
[*string*][;REL]
[ALL | @] [;UNN]

(Default: display previous 20 commands)

(BJ and ,, are short for LISTREDO)

Commands are numbered sequentially from one as entered; the last 1000 are retained. You can display a single command, a range of commands, all 1000, or all the commands whose name matches the string. You can print the commands with ABSolute line numbers (the default), RELative line numbers (-5/-4), or UNNumbered. You can write the commands to your terminal or OUT to a temporary file. If you want to redo any of these commands, see Do, Redo, and Before.

Examples

```
$listredo 5
$listredo 5/10
$listredo help                          {print all Help commands}
$listredo -10                           {print last ten commands}
$listredo ALL                           {print entire redo stack}
$listredo @;rel                         {print ALL, relative numbers}
$listredo purge                         {print all Purge commands}
$listredo purge xx                      {print all "purge xx" commands}
$listredo @purge                        {print all with "purge" anywhere}
$listredo 1/10;out=*lp                  {dump commands to printer}
$listredo @;unn;out=save                {write commands to a file}
```

Notes

The Listredo command cannot be abbreviated, but BJ and ,, (comma comma) are accepted as a short forms.

Persistent Redo

Redo commands can be saved in a permanent file and can therefore be used from another session. You can use the **Set redo** command to specify a filename to save your redo commands. Please see the Set Redo command for details.

Output Command [O]

Specifies the Output file.

OUTPUT * | *filename* [TEMP] [ERASE]

By default, the Output file is permanent and named "Output". If you specify Columns None, the output file will have variable-length records. When Columns Fixed is specified, STExport creates the output file with fixed-length records.

STExport computes the maximum record size by computing the maximum length of each formatted field and adding them together. If the heading line is longer than this, the length of the heading line is used as the record length. You can use a :file equation to reduce the record size, but it is not recommended since too short a record size can cause STExport to fail in the middle of a task.

File Equations

STExport allows the use of file equations to help ensure a file is built with the attributes that you require in the output file.

One example of this is when you are generating HTML output and want to use it with the Apache Web server on MPE. The Web server needs the files to be bytestream files.

One caveat for this technique with bytestream files is that you have to specify a disc value greater than or equal to the total number of bytes that will be written to the file, because by definition a bytestream file has a record size of only one byte. An example of this technique is as follows:

```
:file x=myhtml;rec=,,b;disc=100000
$in mysdfile
$html table title "My Web Page"
$out *x
$exit
```

Stdlist

If the output file name is *, each output record is written to stdlist. This is useful for trying out different formatting combinations until you find the one that best fits the application that you want to import data into. For example,

```
$input     sdfile
$output    *
$xeq
$floating  fixed 2                    {change one option}
$input     sdfile
$output    *
$xeq                                  {view the result}
$sign      none                       {change a different option}
$input     sdfile
$output    *
$xeq                                  {and so on}
```

Quote Command [Q]

Specify which quote character, if any, is to be used around byte-type fields.

QUOTE NONE | DOUBLE | SINGLE

(Default: Double)

Most software packages expect byte-type fields to be in one of two formats:

1. Fixed-column (see the Column command).
2. Surrounded by single- or double-quotes. In this case, you may also need to remove trailing spaces (see the Spaces command).

No Quotes for Fixed Columns

Use Quote None to cause byte-type fields to be output as a group of characters. In many cases, you would combine this option with Columns Fixed.

Single or Double Quotes

By default, all byte type fields are surrounded by double quotes. Specify single quotes using the Single option. If a byte type field contains the quote character specified in the quote command, it is replaced with a space. For example, if the input was:

```
customer's
```

and Quote Single had been specified, then the output would be:

```
'customer s'
```

Redo Command [REDO]

Enables you to modify and repeat any of the previous 1000 command lines.

REDO [*start* [/ *stop*]]
[*string*]
[ALL | @]

(Default: redo the previous command)

The Redo command allows you to modify the commands before it executes them. If you do not need to change them, use the Do command. Commands are numbered sequentially from one as entered; the last 1000 are retained. Use the :Listredo command to display the previous commands. You can Redo a single command, a range of commands, or the most recent command whose name matches a string.

The Redo command uses MPE-style editing logic (D, I, R, U and >). The default mode is to replace characters. To delete, type DDDD under the characters to be removed. To insert, type I under the insertion spot, then the new characters. To undo your changes, type U. To append to the end of the line, use >xxx. To delete from the end of the line, use >DD. To replace at the end of the line, use >Rxxx. And to erase the rest of the line, use D>. If you prefer Qedit-style editing (Control-D, etc.), use the Before command instead of the Redo command.

Examples

```
$listf @.soruce                          {'source' is not spelled right}
NON-EXISTENT GROUP.  (CIERR 908)
$redo                                    {redo most recent command}
listf @.soruce                           {last command is printed}
         our                             {you enter changes to it}
listf @.source                           {edited command is shown}
                                         {you press <return> }

$listredo all
$redo 5                                  {redo 5th command in stack}
$redo                                    {redo previous command}
$redo -2                                 {redo command before previous}
$redo 8/10                               {redo 8th through 10th}
$redo -10/                               {redo -10 through last}
$redo purge                              {redo last Purge command}
$redo purge temp                         {redo last "purge temp"}
$redo @temp                              {redo last containing "temp"}
```

Notes

The Redo command cannot be abbreviated.

To save more commands, use a :File command on the file Stexredo before you run STExport:

```
file stexredo;disc=5000
run stexport.pub.robelle
```

Reset Command [R]

Cancel the current task.

RESET

Reset closes the current Input file, then resets the Output file name to "Output". Formatting options are not reset, only the task-related commands are reset. If you try to reset an individual command, STExport prints a warning.

Set Command [S]

Enables or disables certain operating options within STExport. These options are not reset by Xeq or Reset commands.

```
SET     [CLEANCHAR <string> ]
        [MAPPED         ON|OFF]
        [REDO  filename]
        [STATISTICS     ON|OFF]
        [VARSUB         ON|OFF]
        [VARSUBDEBUG ON|OFF]
        [WARNINGS   ]  ON|OFF
        [XMLTAGCHAR ] "." | "_" | "-"
        [ZONEDFIX] ON|OFF
```

CleanChar

SET CleanChar <string>

(Initially: set to space)

Sets what character is used to replace a Clean character. If you just want to clean the fields and replace the "bad" character with no character then you need to set the CleanChar in the following manner:

```
>Set CleanChar "<null>"
```

See the Clean command for more details.

Mapped

SET MAPPED ON | OFF

(Initially: OFF)

Mapped forces STExport/iX to read the input file using mapped file access. Specifying this option is an error in STExport/V. If the input is not in memory, the wall time performance is worse with Set Mapped On, but CPU time performance is better. You must Set Mapped On before specifying the input file.

Redo

SET REDO *filename*

(Initially: unnamed temporary file)

Commands entered at the STExport prompt are saved in something called the redo stack. You can recall commands from the redo stack by using other commands such as Before, Do and Redo. By default, the redo stack is stored in a temporary file and discarded as soon as you exit. This temporary stack is not preserved across STExport invocations.

The new Set Redo command assigns a permanent file as the redo stack, allowing the stack to become available for future STExport invocations. For example, to assign the Myredo file as a persistent redo stack, enter

```
$Set Redo Myredo
```

If the file does not exist, STExport creates it. Otherwise, STExport uses the existing file. All subsequent commands are written to the persistent redo stack. The setting is valid for the duration of the STExport session. As soon as you exit STExport, the setting is discarded. Next time you run STExport, you will get the temporary stack.

If the file name is not qualified, the redo stack is created in the logon group and account. This may be desirable if you want to have separate stacks. If you want to always use the same persistent stacks, you should qualify the name.

The Verify command shows which stack is currently in use. If it shows <temporary>, then STExport is using the default stack. Anything else is the name of the file used on the Set Redo command.

Concurrency

When STExport uses the default temporary stack, it is only accessible to that particular instance of STExport. You can run as many STExport instances as you need and each one gets its own redo stack. With temporary stacks you will never get into concurrency problems.

If you start using a persistent redo stack, however, you might start running into concurrency problems. A persistent redo stack can only be used by one STExport instance at a time. If you try to use a persistent redo stack that is already in use, you will get the following message:

```
$Set Redo Myredo
EXCLUSIVE VIOLATION: FILE BEING ACCESSED  (FSERR 90)
Unable to open file for REDO stack
```

In this situation, STExport continues to use the redo stack active at the time and lets you continue working as normal.

Qedit can also have permanent redo stacks. To prevent products from writing to each other's redo stack, it is advisable to have separate stacks for each product by giving them different file names. For example, if you use the command

```
set redo myredo
```

you will have a redo stack called Myredo for your STExport commands. If you exit STExport, then run Qedit and supply the same command Set Redo command, your Qedit commands will be written to the same file that was used for your STExport commands.

Statistics

SET STATISTICS ON | OFF

(Initially: OFF)

Statistics causes STExport to print statistics at the end of each task.

Varsub

SET VARSUB ON | OFF

(Initially: OFF)

Setting Variable Substitution causes STExport to resolve any CI variables in a command before processing.

VarsubDebug

SET VARSUBDEBUG ON | OFF

(Initially: OFF)

Suprtool, now has a setting called Set VarsubDebug on which will print out the line after the variable substitution has occurred. This setting only works if Set Varsub is on and Set VarsubDebug is on.

```
setvar outfile &
:"/GREEN/SUPRTEST/filename901234567890123456789012345678901234567890123
45678901"
:run suprtool.pub.robelle
SUPRTOOL/iX/Copyright Robelle Solutions Technology Inc. 1981-2007.
(Version 6.2 Internal)  TUE, OCT 30, 2007,  2:58 PM  Type H for he
>set varsub on
>set varsubdebug on
>in file1sd.suprtest
vd:in file1sd.suprtest
>output !outfile,link,temp
vd:output /GREEN/SUPRTEST/filename90123456789012345678901234567890123
vd:2345678901,link,temp
```

The output is formatted into 74 byte chunks and printed with a preceding “vd:” so the “substituted” line is clear.

Warnings

SET WARNINGS OFF

(Initially: ON)

Suprtool normally prints warning messages out to $stdlist. You can turn off these messages when you are running from batch by issuing a Set Warnings off command. If you are simulating batch mode with the Set Interactive Off command, you must do the Set Warnings off after the Set Interactive Off.

The default for this setting is On.

Xmltagchar

SET XMLTAGCHAR "." | "-" |"_"

(Initially: ".")

In XML the tags that surrounded the data can not have any special characters other than hyphen, underscore and period ("-","_", "."). So STExport replaces any of the invalid special characters with a "." by default. You can change the default character To be something else with the following set command:

```
  $Set xmltagchar "_"
```

STExport will only allow the hyphen, underscore and period to be set with this command.

ZonedFix

SET ZONEDFIX OFF

(Initially: OFF)

Set ZonedFix has been added to fix a zoned field that when converted to byte, results in "?", meaning that the source field has some characters that were not expected in a Zoned field. If Set ZonedFix is On STExport zeroes out the field after an attempted conversion of the field yields characters that cannot be converted.

The default for this setting is Off.

Sign Command [SI]

Specify what should be done with the sign character for numeric-fields.

SIGN NONE | FLOATING | LEADING | TRAILING

(Default: Floating)

All numeric-type fields, except logical fields, have a sign. Integer and floating-point fields can have either a space " " (for positive values) or a negative sign "-". Packed- and display-type fields can have a space " " (neutral), a plus sign "+" (for positive values), or a negative sign "-".

Specify Sign None to cause STExport to completely ignore the sign. If you specify Sign None, no error or warning message appears if any numeric-types have a negative value.

Leading vs. Floating

If you specify Columns Fixed, it is easy to see the difference between a leading versus a floating sign. A leading sign always appears in the same column whereas a floating sign always appears before the first digit of a number. For example,

```
Sign Leading            Sign Floating
-   22415               -22415
-     207                 -207
-   16600               -16600
-   21910               -21910
-    8411                -8411
-      42                  -42
-   16713               -16713
-    7970                -7970
```

Trailing Sign

Specify Sign Trailing to cause the sign character to appear after each formatted number. Remember that for many numeric-types the sign for positive numbers is a space. STExport always leaves room for the sign, even if it is a space.

Spaces Command [SP]

Specify whether trailing spaces are to appear in byte-type fields.

SPACES NONE | TRAILING

(Default: None)

If byte-type fields are surrounded with quotes (see the Quote command), the Spaces command determines whether trailing spaces in byte-type fields appear within the quotes. Use Spaces Trailing if you want to retain all of the spaces in a byte-type field.

Software packages that store variable-length character data treat trailing spaces as data. Use Spaces None to remove trailing spaces for data that is imported into these applications.

Use Command [U]

Specifies a file of commands to be executed as a group.

USE[Q] *filename*

Examples

A usefile makes your task easier by allowing common commands to be specified once in an external file. For example, the following usefile contains all the commands for creating the Invcust file:

```
$use invuse
input    invoices                    {input file to format}
floating fixed 2                     {formatting option}
output   invdata                     {produces the file we want}
exit
```

STExport prints the lines in the usefile, including the comment lines. This allows you to include instructions and reminders in the usefile. In the example above, there were no commands for the user to enter.

Notes

Usefiles cannot be nested in STExport. The usefile may be any unnumbered text file or a Qedit workfile, but no more than 256 characters per record are processed.

By default, STExport displays the commands in a usefile as they are executed. STExport can execute commands *quietly* using the Useq command. For compatibility with Qedit, Useq can be abbreviated to UQ.

Verify Command [V]

Print the definition of the current task.

VERIFY

Verify prints the current Input and Output files and all export specifications; in other words, it is a Verify All command.

Xeq Command [X]

Perform the current task.

XEQ

Xeq checks that you have specified an Input file and an Output file. Then it performs the task and creates the Output file. Finally, it closes the files, ready for you to specify another task or Exit. If you also wish to leave STExport after completing the task, use Exit instead of Xeq.

XML Command [XML]

Use XML to produce XML Documents for either Internet or Intranet applications.

XML

VERSION |2"string"|

DOCTYPE |2"string" &| filename|

FILE |2"string"|

RECORD |2"string"|

Example

STExport can generate "well-formed" XML output with just a few commands.

```
$input file1sd
$xml
$output myxml
$xeq
```

These four simple commands will generate the following file that packages such as XMLSpy will consider to be "well-formed" XML. The result of such an STExport task will look as follows:

```
<?xml version='1.0'?>
<file>
<record>
<CITY>Los Altos</CITY>
<CREDIT-RATING>100000</CREDIT-RATING>
<CUST-ACCOUNT>4003302</CUST-ACCOUNT>
<CUST-STATUS>20</CUST-STATUS>
<NAME-FIRST>Ralph</NAME-FIRST>
<NAME-LAST>Perkins</NAME-LAST>
<STATE-CODE>CA</STATE-CODE>
<STREET-ADDRESS>Room 655</STREET-ADDRESS>
<STREET-ADDRESS>Los Altos     040033022</STREET-ADDRESS>
<ZIP-CODE>93002</ZIP-CODE>
</record>
</file>
```

Notes

By default, STExport will add the simplest version tag at thebeginning of the file, then it inserts a <file> and matching </file> at the beginning and the end of the file. Then STExport encloses each record from the input file in a <record> and </record> tag. Finally, the Self-Describing tags are added around each field's data values and edited appropriately.

Naturally users would want options to customize and specify the various options and tags themselves, in order to generate a file that is acceptable to their tools.

Version

You can specify the "version" tag at the beginning of the XML file with the following command:

```
xml version "?xml version='1.0' encoding='ISO-8859-1'?"
```

STExport will put the "<" and ">" around what is specified in the version string.

Doctype

A Document Type Declaration can be made at the beginning of the file via the !DOCTYPE specification. This typically tells whatever tool that is parsing the xml file where the DTD for the file resides.

In STExport you can specify simple one-line doctype specs with the following command:

```
xml doctype '!DOCTYPE address-book SYSTEM "address-book.dtd"'
```

This will write the doctype specification at the top of the output file, directly after the XML version specification.

More complicated and lengthy doctype specifications can be implemented by using the filename option, whereby STExport will append the contents of a named file directly after the version specification. The syntax can be simply:

```
xml doctype myfile.xml
```

File

You can customize the "file" tags with the following command commands:

```
xml file "orders"
```

STExport will put the "<" and ">" around what is specified in the File string.

Record

You can customize the "record" tags with the following command commands:

```
xml record "Details"
```

STExport will put the "<" and ">" around what is specified in the Record string.

Example

You can enter multiple XML commands per task to set the XML options you require.

```
$in file1sd
$xml version "?xml version='1.0' encoding='ISO-8859-1'?"
$xml file "Orders" record "Details"
$out myfile
```

An example of the output generated by the above commands is as follows:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<Orders>
<Details>
<CITY>Los Altos</CITY>
<CREDIT-RATING>100000</CREDIT-RATING>
<CUST-ACCOUNT>4003302</CUST-ACCOUNT>
<CUST-STATUS>20</CUST-STATUS>
<NAME-FIRST>Ralph</NAME-FIRST>
<NAME-LAST>Perkins</NAME-LAST>
<STATE-CODE>CA</STATE-CODE>
<STREET-ADDRESS>Room 655</STREET-ADDRESS>
<STREET-ADDRESS>Los Altos    040033022</STREET-ADDRESS>
<ZIP-CODE>93002</ZIP-CODE>
</Details>
</Orders>
```

Tags

In XML the tags that surrounded the data can not have any special characters other than hyphen, underscore and period ("-","_", "."). So STExport replaces any of the invalid special characters with a "." by default.

You can change the default character to be something else with the following set command:

```
$Set xmltagchar "_"
```

STExport will only allow the hyphen, underscore and period to be set with this command.

Quotes

All of STExport's XML command options (version, doctype, file and record) allow for a string to be passed via surrounding quotes. The quotes may be either single or double, but keep in mind that if the string is to contain double quotes, then you should surround the entire string with single quotes.

Zero Command [Z]

Specify whether leading zeros are to appear in numeric fields.

ZERO NONE | LEADING

(Default: None)

Use Zero None to force all numeric fields to have leading zeros removed. If a numeric field has implied decimal places, STExport always formats the number with at least one digit to the left of the decimal place, even if it is zero.

Use Zero Leading to force all numeric fields to be zero-filled. In this case, Sign Leading and Sign Floating both cause the sign to appear in the same place (in front of the leading zeros).

Example of STExport Output

Example

In this example we show you how to use Suprtool and STExport. We start with an IMAGE/SQL dataset, identify the fields that are dates and the number of implied decimal places in other fields. We then produce a self-describing file using the dataset as input and show the default output from STExport.

We also show you how easy it is to migrate data from MPE files and databases to an Oracle database.

The Form command displays the fields in a dataset or a self-describing file. For files, this information is stored in the user labels of an MPE file and is not accessible with other tools. Use the Form command to obtain the record layout of STExport input files.

Sales File

We will be formatting data from an IMAGE/SQL sales dataset that has the following form:

```
>form d-sales
Database: STORE.DEMO.ROBELLE

D-SALES          Detail    Set    5
    Entry:                    Offset
       CUST-ACCOUNT          Z8       1  (!M-CUSTOMER)
       DELIV-DATE            J2       9
       PRODUCT-NO            Z8      13  (M-PRODUCT)
       PRODUCT-PRICE         J2      21
       PURCH-DATE            J2      25
       SALES-QTY             J1      29
       SALES-TAX             J2      31
       SALES-TOTAL           J2      35
    Capacity: 602 (14)  Entries: 8  Highwater: 8  Bytes: 38
```

Dates and Decimal Places

We use Suprtool's Item command to identify which of the fields in the d-sales dataset are dates and which fields have implied decimal places:

```
>item deliv-date    ,date    ,yyyymmdd
>item purch-date    ,date    ,yyyymmdd
>item product-price ,decimal ,2
>item sales-tax     ,decimal ,2
>item sales-total   ,decimal ,2
```

Salefile

We now produce a file called "salefile" using Suprtool's Output,Link option. The Link option produces a self-describing file, complete with the date and decimal-place information:

```
>get d-sales
>output salefile,link
>xeq

>form salefile

File: salefile           (SD Version B.00.00)
  Entry:                    Offset
     CUST-ACCOUNT          Z8      1
     DELIV-DATE            I2      9     <<YYYYMMDD>>
     PRODUCT-NO            Z8     13
     PRODUCT-PRICE         I2     21     << .2  >>
     PURCH-DATE            I2     25     <<YYYYMMDD>>
     SALES-QTY             I1     29
     SALES-TAX             I2     31     << .2  >>
     SALES-TOTAL           I2     35     << .2  >>
Limit: 108  EOF: 8  Entry Length: 38  Blocking: 107
```

Notice how the Form command correctly identifies which fields are dates and which fields have implied decimal places. STExport uses this information to format the file.

STExport Output

We then use STExport to read the self-describing Salefile to produce our sample output on stdlist. To demonstrate how dates are handled, we insert a separator in each date field:

```
:run stexport.pub.robelle
$input salefile                          {self-describing input file}
$date  yyyymmdd "-"                      {dates with a dash separator}
$output *                                {output to stdlist}
$xeq
10020,1993-10-05,50511501,98.31,1993-10-01,0.02,27.53,224.15
10003,1993-10-15,50511501,98.31,1993-10-15,0.01,13.76,112.07
10003,1993-10-15,50512501,145.62,1993-10-15,0.01,20.39,166.00
10003,1993-10-15,50513001,192.20,1993-10-15,0.01,26.91,219.10
10016,1993-10-21,50521001,24.59,1993-10-21,0.03,10.33,84.11
10016,1993-10-21,50532001,139.85,1993-10-21,0.01,19.58,159.42
```

There are no byte-type fields in the input file, so all fields are converted from their internal numeric representation to a string of digits. All date fields were converted from their internal yyyymmdd format to the external yyyymmdd format with a dash separator between day, month, and year. All fields with implied decimal places have been converted with a decimal point.

Load Data Into Oracle

If you need to load the export file into an Oracle database, you can use Oracle's own SQL*Loader. Files created with STExport can be processed immediately with SQL*Loader. Suppose we want to load the data extracted in Salefile. We would use STExport to format the information and store the results in Expsale.

```
:run stexport.pub.robelle
$input salefile                          {self-describing input file}
$date  yyyymmdd "-"                      {dates with a dash separator}
$output expsales                         {output to a file}
$xeq
```

If the Oracle database resided on an HP 9000, we would need to transfer Expsale. How you transfer the file is not important as long as it gets there in the same format, including line separators. Also note that the file should have an extension. SQL*Loader expects a .dat extension by default.

The Oracle table should have all the necessary columns. To create a table with the fields in Salefile, you could use the following:

```
create table sales_details (
       CUST_ACCOUNT  dec(8),
       DELIV_DATE    date,
       PRODUCT_NO    dec(8),
       PRODUCT_PRICE dec(8,2),
       PURCH_DATE    date,
       SALES_QTY     dec(6),
       SALES_TAX     dec(8,2),
       SALES_TOTAL   dec(8,2)
       ) tablespace USERS;
```

SQL*Loader requires what is known as a control file. It contains the load specifications such as the data file, the destination table, the field delimiter, the text field delimiters and the column specifications. In this case, the control file (expsales.ctl) looks like this:

```
   load data
   -- Specify input datafile.  dat extension is assumed.  --
   Infile expsales

   -- Name of the table where the data is loaded.  --
   -- Append new rows to existing data, if any.   --
   Append Into Table sales_details

   -- Fields are separated by commas --
   Fields Terminated By ','

   -- Character fields are enclosed in double-quotes --
   Optionally Enclosed By '"'

   -- Specify the column names as they appear          --
   -- in the data records.                             --
   -- For a Date-type column, specify the input format --
   -- Example:  column Date "YYYYMMDD"                 --
   (CUST_ACCOUNT,
    DELIV_DATE    date "YYYYMMDD",
    PRODUCT_NO,
    PRODUCT_PRICE,
    PURCH_DATE    date "YYYYMMDD",
    SALES_QTY,
    SALES_TAX,
    SALES_TOTAL)
```

In its simplest form, SQL*Loader is invoked with the following command:

```
$ sqlload userid=username/password
          control=expsales.ctl
          log=expsales.log
```

The *username* and *password* are valid Oracle connect information. SQL*Loader reads the load specifications from the file specified in the *control* keyword. It writes operation information and statistics to the file specified in the *log* keyword. It also creates a number of files to report data problems etc. SQL*Loader has many other

options to control the load task. Refer to the appropriate Oracle documentation for details.

Limits Within STExport

Maximums

To achieve maximum speed, STExport/V stores all the necessary information in the HP e3000 stack. This limits the size of input and output records and restricts the number of fields per file. Users should be aware of these limits before attempting to use STExport with extremely complicated databases. STExport/iX stores information in a simulated HP e3000 stack, so all the limits apply to both STExport/V and STExport/iX.

Delimiter - Maximum Length - 3 Bytes

The delimiter must appear between every field in the output record. To help avoid exceeding the maximum output record length, the maximum delimiter length is three characters.

Input File - Maximum Record Size - 8192 Bytes

We recommend that you use Suprtool's Extract command to minimize the input record size.

Input File - Maximum Block Size - 8192 Bytes

By default, Suprtool restricts the maximum block size to 4,096 bytes. You can use the Set Blocksize command to increase this size up to 16,384 bytes (8192 words). If you increase the maximum blocksize, it is likely that Suprtool will produce an output file that STExport cannot read.

Input File - Maximum Fields - 255

If you must have more than 255 fields, use Suprtool's Define and Extract commands to extract several fields as one contiguous series of bytes.

Output File - Maximum Record Size - 8192 Bytes

When formatting many fields, it is possible to produce large output records. Once again, using the Extract command to minimize the size of the input records will avoid large output records.

The total length of the Heading line in the output file is also restricted to 4096 bytes.

Welcome to RPort

Welcome to RPort

Welcome to RPort for MPE Version 6.2. RPort converts fields in a self-describing input file into an output file that can be imported into different applications.

Summary of the RPort commands:

Abort	Heading	Sign	= expression
Before	Help	Size	:OS Command
Comma	Hide	Subtotal	
Decimal	Input	Title	
DO	Listredo	Total	
DOLlar	Output	Use	
Exit	Redo	Verify	
Floating	Reset	Xeq	
Form	Set	Zero	

Installing RPort

RPort is installed as part of the Suprtool installation process. See the "Installing Suprtool" chapter of the *Suprtool User Manual* for more details on how to install both Suprtool and RPort.

Built-In File Names

RPort requires an external file for the Help command. RPort dynamically changes this file name, but you can use a :File command to override RPort's choice.

Default File Names

If you are running RPort on MPE V/E or MPE/iX, RPort finds the name of the RPort program file name (e.g., rport.Pub.Robelle). RPort uses this name to determine the name of the other built-in file names. If RPort cannot call the procinfo intrinsic (e.g., this intrinsic doesn't exist on MPE V/R), it assumes you are running rport.Pub.Robelle.

Account Name

The account name for all built-in files is the same as the one where RPort is running (e.g., if you :run rport.Pub.Dev, the help file for RPort is rport.Help.Dev).

Group Name

RPort examines the group name where RPort is running to determine the group name for the built-in file names. If the group name is Pub, RPort assumes help files are in the Help group. The same assumption is made for any group name where the first three letters are not Pub. When RPort is run from a group that starts with Pub (e.g., Pub), RPort assumes the help files have the same suffix (e.g., help).

Examples

Here are a few examples of the names that RPort would use for the RPort help file:

RPort Program File Name	RPort Help File Name
rport.Pub.Robelle	rport.Help.Robelle
rport.Pub.Robelle	rport.Help.Robelle
rport.Pub.Account	rport.Help.Account
rport.PubRob.Account	rport.HelpRob.Account
rport.Robelle.Dist	rport.Help.Dist

Changing Built-In File Names

You can use a :File command to tell RPort where the help file is located. Your :File command must use the file name that RPort dynamically assigned to the help file. For example, if RPort is called RPort.Robelle.Dist, you would use this :File command:

```
:file rport.help.dist=stephelp.robelle.dist
```

Accessing RPort

How To Run RPort

To access RPort, type the following command:

```
:run rport.pub.robelle
RPort/Copyright Robelle Solutions Technology Inc. 1996-2001
(Version 6.1)
%
```

After a short pause, RPort takes over your terminal and prints out some identifying information. You will notice that your command prompt has changed to "%", telling you that you have made it into RPort. RPort expects you to type command lines, ending each one with Return.

How to Xeq an RPort Task

Normally, you enter a series of commands. These commands specify the Input file, the Output file, and the formatting options. Finally, you enter an Xeq or an Exit command. This begins the actual RPort task.

If you entered the Exit command, RPort finishes the current task, then returns you to the operating system or the program that ran RPort.

```
%EXIT
```

If you entered the Xeq command, RPort finishes the current task, then prompts you for another task. This continues until you enter the Exit command. If you wish to terminate RPort immediately (perhaps you are confused), enter Exit Abort. This terminates the RPort program immediately, without attempting any task.

Son Process

If you run RPort within Qedit or Select, you can retain the RPort process and quickly re-activate it later.

```
:run qedit.pub.robelle
/:r rport.pub.robelle
%...
%exit
RPORT.PUB.ROBELLE is still alive. Okay to HOLD ? Y
/...
/:activate
RPORT.PUB.ROBELLE
%...
%exit
Program Held.  Use :ACTIVATE/:RUN;HOLD to re-run.
/...
```

Suprtool Rport Command

This feature is not available yet in Suprtool.

You can access RPort from within the Suprtool program using Suprtool's Export command. Suprtool then runs RPort for you. All of RPort's commands are available to you in Suprtool. When passing RPort commands through Suprtool, you must preface each RPort command with *Rport.* For example,

```
:run suprtool.pub.robelle
>base customer
>get m-customer
>sort cust-no
>output invoices,link
>xeq
>rport input invoices
>rport sign none                    {send "sign none" to RPort}
>rport output invfile
>rport exit                         {exit from RPort (not Suprtool)}
```

Refer to the Rport command in the *Suprtool Commands* chapter of the Suprtool user manual for more information.

Preventing MPE Commands

If you want to prevent RPort from executing any MPE commands (e.g., :Purge), you Run RPort with Parm=8192. This feature is automatically invoked by Suprtool's Rport command, when Set Limit MPE Off has been specified inside Suprtool.

```
:run rport.pub.robelle;parm=8192
```

Exit with Verify

Some users find that they Exit from RPort inadvertently. For RPort to get user approval on Exit, Run RPort with Parm=64.

```
:run rport.pub.robelle;parm=64
%e
Okay to exit [no]:
%
```

Preventing RPort from Suspending

If you run RPort from within HPDesk (and some other programs), RPort suspends on Exit but HPDesk does not notice. The next time you run RPort you will get a new copy of RPort. Eventually you will have many suspended copies of RPort hanging

from HPDesk, consuming system resources. Running RPort with Parm=32 forces RPort to terminate on Exit rather than suspend.

```
:run rport.pub.robelle;parm=32
```

Job Control Word

RPort sets the system job control word (JCW) to a fatal state when RPort fails in a batch job. RPort sets only the high-order bit of the JCW. That is, it adds 32,768 to the existing JCW value. HP subsystems use the other bits of the JCW, so RPort does nothing to them.

Using RPort in Batch

RPort operates in session mode or batch mode. In batch usage, any "error" message causes RPort to quit, setting the Job Control Word to flush the remainder of the JOB. Warning messages do not cause an abort.

In batch mode, RPort does not prompt for missing information as it does in session mode. Instead, it attempts to choose the alternative that has the least chance of destroying valid data. For example, if the Output file is a duplicate file name in batch mode, RPort saves the new Output file with a "made up" name (OUTPUTnn, where nn is from 00 to 20), prints a warning message, and aborts.

Summary of Parm= Values

Value	Function
32	don't suspend, terminate completely
64	check with user before Exiting
8192	don't allow MPE commands

Values may be combined by adding them together. For example, Parm = 96 means "check with me before exiting, then when I do actually exit, terminate RPort completely instead of suspending".

RportOutCount JCW

When RPort closes the output file, it sets a JCW named RportOutCount with the number of output records. This is the same number reported in the total line (i.e., OUT=). You can use this JCW to control job stream execution by checking if the SprlinkOutCount JCW is non-zero. I there are more than 65,535 output records, RPort sets RportOutCount to 65,535.

You can use the :Showjcw command to see the value of a JCW. For values greater than 16K, Showjcw displays either the word WARN, FATAL or SYSTEM followed by some digits. These words correspond to the following values:

```
WARN 16384

FATAL 32768

SYSTEM 49152
```

Add the value of the word to the number that appears with it for the true value of RportOutCount. For example,

```
:showjcw Rportoutcount

RPORTOUTCOUNT = WARN8616

16384 + 8616 = 25000
```

The MPE/iX :Showvar command can also be used to see the value of a JCW. Showvar displays the full, correct number (e.g., 25,000) up to the maximum of 65,535.

RPort also sets two other JCWs: RPORTOutCount1 and RPORTOutCount2. These communicate the full OUT= value to the RPORT interface.

RPORTFullCount Variable

On MPE/iX, when RPort closes the output file, it also sets a variable named RportFullCount with the number of output records. This is the same number reported in the total line (i.e., OUT=). The advantage of the variable is that if more than 65,535 records are written to the output file, the value of the RportFullCount variable is not truncated.

Introduction to RPort

Importing Data

Use RPort to produce a formatted simple formatted report with Titles, Headings Subtotals and Totals.

Input File

RPort reads one input file and formats each input record into one record in the output file. The Input file must be a self-describing file (use the Output-Link option in Suprtool).

Dates and Decimal Places

Use Suprtool's Item command to specify date formats and the number of implied decimal places when you create the self-describing file. RPort uses this information to correctly format the information in the output file. See Appendix A for a complete example of how to use Suprtool's Item command and Output-Link option to create an input file for RPort.

Data-Types

Each RPort formatting command applies to all fields of a specific data-type (i.e., you cannot specify formatting field by field, only by type). For example, all numeric-type fields can be formatted the same.

The main data-types that RPort identifies are

Byte-Type:	RPort assumes that character information is stored in byte-type fields.
Numeric-Type:	The numeric data-types are integer, logical, floating-point, packed, and display. RPort converts the internal representation of each data-type into a string of ASCII digits. By default, all numeric-type fields have a leading sign and are variable length. Where appropriate, numeric-type fields are converted with a decimal point.
Floating-Type:	All commands that affect numeric-type fields also affect floating-type fields. In addition, you can use the Floating command to specify the format and decimal places for floating-type fields (i.e., Classic or IEEE floating-point numbers).
Date-Type:	If a field has a date format, RPort does extra formatting. By default, dates are formatted into yyyymmdd (e.g., 20001125).

Formatting Commands

Use the following table to determine which command applies to which data-type:

Command	Data-Type
Date	date-type
Floating	floating-type
Sign	numeric-type
Zero	numeric-type

CI Variable Substitution

RPort is able to substitute any CI variables from any command line source, whether through interactive, use file or batch input.

In order to use this feature, issue the following Set command:

```
set varsub on
```

Variable Substitution is not on by default for backward compatibility.

Batch

Since the Streams facility (under default setups) will replace any "!" found in the first column of a job stream. Anytime you want to specify an entire line thru Variable Substitution you will need to leave a space before the variable is specified.

```
!setvar input "i dfile by message-no"
!run rport.pub.robelle
%set varsub on
% !input
```

Notes

For MPE commands some variables will be resolved twice when passed off to MPE, which will give different values for a variable.

Setting variables at the CI level:

```
MPEXL:setvar x 10
MPEXL:setvar y "!!x"
MPEXL:showvar [xy]
X = 10
Y = !x
```

Setting variables within RPort

```
%set varsub on
%setvar a 10
%setvar b "!!a"
%showvar [ab]
A = 10
B = 10
```

Because Rport does one level of variable substitution prior to the command being passed off to MPE, setting variables in Rport that involves other variables will give different results from MPE. We recommend that you set the variables prior to running Rport, or that you temporarily turn off variable substitution with the Set Varsub Off command.

```
%set varsub off
%setvar a 10
%setvar b "!!a"
%showvar [ab]
A = 10
B = !a
%set varsub on
```

RPort Commands

General Notes

When you run RPort, it prompts for commands on $stdlist with a "%" character and reads command lines from $stdinx. RPort commands contain a command name followed by one or more parameters, and are patterned after the same commands in Suprtool.

In this chapter, we describe the RPort commands in alphabetical order. Following each command name in brackets is the minimal abbreviation for the command. For example: [I] for Input and [O] for Output.

Abbreviating

You may shorten the command to the first letter of the command name.

```
%v                                  {verify}
%x                                  {xeq}
```

Uppercase or Lowercase

You may enter the letters in either uppercase or lowercase, because RPort upshifts everything in the command line except literal strings within quotes ("abc") and file names. These two commands are identical:

```
%EXIT
%exit
```

Comments on Command Lines

Comments may appear at the end of any command line, when they are surrounded by braces. Many of the examples in this manual show comments at the end of each command line. You can enter a comment as the only item in a RPort command line. When continuing command lines, the comment can appear before or after the continuation character.

```
%{ format reals with two decimal places. }
%input invoices
%floating fixed 2                       {Floating option}
%output invfile                         {produces the file we want}
%exit
```

STREAMX

STREAMX is a product from VESOFT that permits you to build flexible job streams. STREAMX contains a complete programming language with loops, prompts, and parameter substitution. A problem arises when trying to enter comments into a Suprtool batch job that will be submitted with STREAMX. Suprtool uses the {...} pair to delimit comments. STREAMX uses these same characters for expressions.

You cannot change Suprtool's comment character, but you can change the {...} characters in STREAMX. The following example shows how to change the STREAMX expression characters from {...} to ~...~:

```
!job example,user.acct
::setbraces ~~
!run rport.pub.robelle
%input invoices                         {Input file to format}
%floating fixed 2                       {Option for floating-point numbers}
%output invdata                         {Produces the file we want}
%exit
!tell ~hpjobname~,~hpuser~.~hpaccount~;Example done.
!eoj
```

MPE Commands

RPort also accepts MPE commands, with or without a colon.

```
%:comment
%comment
```

For commands that are the same in both RPort and MPE, RPort executes the MPE command only if you type the colon. For example:

```
%help                                   {you get RPort help}
%:help                                  {you get MPE help}
```

MPE/iX Commands

RPort/iX executes any MPE command (e.g., Run), UDCs, and command files. **Caution:** programs that suspend, instead of terminating, are not killed by the HPCICOMMAND intrinsic.

File Names

RPort's Input and Output commands accept a file name in either MPE or POSIX format. File names starting with "/", "./", or "../" are treated as POSIX file names. All other file names are assumed to be MPE file names. MPE file names are upshifted and POSIX file names are not. POSIX file names are limited to a maximum of 240 characters. Here are some examples of POSIX file names:

```
:hello david,mgr.dev,david
:CHDIR SUBDIR
:run rport.pub.robelle
%input ./file
%verify input
/DEV/DAVID/SUBDIR/file
```

Calculator

Any command line beginning with an equal sign (=) is treated as a calculator expression. This feature can be used to do other calculations without the need of an electronic calculator.

You can obtain a short description of the calculator by entering the following:

```
=?                                          {? gives a summary of = functions}
```

For a detailed description of the calculator and its options, see the Suprtool manual.

Control-Y

You can interrupt a RPort task with the Control-Y key (hold down Control while striking Y). RPort responds by telling you how far it has gotten (IN=, OUT=, etc.), and asking if you wish to stop. Hit the Return key to continue or type `YES` to stop the task.

Abort Command [A]

Abort to exit out of Rport.

ABORT

(Default: Abort Rport)

The Abort command will exit you out of Rport, it is helpful to exit out of Rport and NOT execute any of the commands that may have been entered.

Before Command [B]

Repeat any combination of the previous 1000 command lines, with or without editing.

BEFORE [*start* [/ *stop*]]
[*string*]
[ALL | @]

(Default: redo previous line)

(BQ=redo without change)

The Before command allows you to modify the commands before it executes them. If you don't need to change them, use BQ or Do.

The Before command uses Qedit-style control characters for modifying the commands. The default mode is to replace characters. To delete, use Control-D; to insert, use Control-B. If you prefer HP-style modify (D, R, I, and U), use the Redo command instead of Before.

Examples

```
%listf @.soruce                          {'source' is not spelled correctly}
NON-EXISTENT GROUP.  (CIERR 908)
%Before                                  {redo most recent command}
listf @.soruce                           {last command is printed}
         our                             {you enter changes to it}
listf @.source                           {the edited command is shown}
                                         {you press Return}

%listredo -10/
%before 5                                {redo 5th command in stack}
%bef 8/10                                {redo 8th through 10th}
%b listf                                 {redo last Listf command}
%b listf temp                            {redo "listf temp" command}
%b @temp                                 {redo last containing "temp"}
%before -2                               {redo command before previous}
%before -5/-2                            {redo by relative lines}
```

Modify Operators

If you wish to change any characters within the line, the modify operators are the regular Control Codes used in Qedit:

- Any printing characters replace the ones above.
- Control-D plus spaces deletes columns above.
- Control-B puts you into "insert before" mode.
- Control-A starts appending characters at the end of line.
- Control-A, Control-D, plus spaces, deletes from the end.
- Control-T ends Insert Mode, allowing movement to a new column.
- Control-G recovers the original line.
- Control-O specifies "overwrite" mode (needed for spaces).

Persistent Redo

Redo commands can be saved in a permanent file and can therefore be used from another session. You can use the **Set redo** command to specify a filename to save your redo commands. Please see the Set Redo command for details.

Comma Command [Co]

The Comma command tells Rport to put commas in a numeric field when it is converted to ascii.

Comma [*fieldname* [, *fieldname*]]

(Default: No comma

You must specify the comma command after the input file so Rport can find the fieldname specified. You can specify a sweries of fieldnames separated by commas.

```
%comma balance,amount
```

Date Command [DA]

Specify a specific date-format for all dates.

DATE NONE | *date-format* ["*separator*"]

INVALID ASTERISKS | NULL | "string"

(Default: yyyymmdd Invalid Asterisks)

Use the Date command to specify an output format for dates. Use the Invalid option to specify how invalid dates should be formatted in the output file. The advantage of the None option is that it formats all dates, whether they are valid or not. If you select a *date-format*, the default Invalid option replaces invalid dates with asterisks "*".

RPort must know which fields are dates and the format of each date. Use Suprtool's Item command and Output,Link option to specify the date information.

Date Format

The *date-format* can be one of:

- ccyymmdd
- yyyymmdd
- ddmmyyyy
- mmddyyyy
- yymmdd
- ddmmyy
- mmddyy
- aammdd

RPort converts each date field from its internal date format into the format that you specify.

Separator Character

By default, RPort formats all dates without a separator between the day, month, and year. Specify your own separator by enclosing it inside quotes after you specify the date format. The separator must be one character long. For example, to specify dates in ddmmyyyy format with a slash separator, use

```
%date ddmmyyyy "/"
```

To specify dates in yymmdd format with a dash separator, use

```
%date yymmdd "-"
```

Invalid Dates

By default, all invalid dates are formatted as asterisks. RPort treats any date that does not have a valid century, year, month, or combination (e.g., February 29, 2000) as invalid. You can specify how you want RPort to format invalid dates by using the Invalid option of the Date command.

If you specify,

```
%date invalid null
```

RPort will produce a zero-length field if you specify Column Variable and spaces if you specify Column Fixed. If you want to specify an explicit string for all invalid dates, do so after the Invalid option. For example,

```
%date invalid "%%%%%"
```

will cause RPort to produce a string of five percent signs for any invalid date.

Example

First, use Suprtool to create the input file with the appropriate date attributes:

```
>get    d-sales
>item   deliv-date,date,mmddyyyy
>item   purch-date,date,mmddyyyy
>output dsales,link
>xeq
```

Then use RPort to read the dsales file. Specify Date ddmmyyyy "-" which causes all valid dates to be formatted in day-month-year format with a dash as the separator:

```
%input  dsales
%date   ddmmyyyy "-"
%output dexport
%xeq
```

Decimal Command [DEC]

Specify the format for the decimal place in numeric fields.

DECIMAL PERIOD | COMMA

(Default: Period)

The fields in the input file must have been created with decimal places, using Suprtool's Item command.

The Decimal command specifies what separator will be used to indicate the decimal place in numeric fields. In North America, the custom is to indicate the decimal place in numbers with a period (.). Outside North America, the custom is to indicate the decimal place with a comma (,). If the decimal place indicator is incorrect, it is harder to import files into other applications.

Do Command [DO]

The Do command repeats (without changes) any of the previous 1000 commands.

DO [*start* [/ *stop*]]
[*string*]
[ALL | @]

(Default: repeat the previous command)

Commands are numbered sequentially from one as entered; the last 1000 of them are retained. Use the :Listredo command to display the previous commands. You can repeat a single command (`do 5`), a range of commands (`do 5/10`) or the most recent command whose name matches a string (`do list`). If you want to modify the commands before executing them, use Redo or Before.

Examples

```
%listredo
%do                                    {do previous command again}
%do 39                                 {do command line 39 again}
%do 5/8                                {do command lines 5 to 8 again}
%do input                              {do most recent Input command}
%do -2                                 {do command before previous}
%do -7/-5                              {do by relative line number}
%do 5/                                 {do command lines 5 to "last"}
```

Notes

The Do command cannot be abbreviated.

Persistent Redo

Redo commands can be saved in a permanent file and can therefore be used from another session. You can use the **Set redo** command to specify a filename to save your redo commands. Please see the Set Redo command for details.

Dollar Command [DOL]

Specify the format for the dollar sign to appear in a numeric field

DOLLAR FLOATING|NONE *fieldname* (Default: None)

The Dollar command allows you to tell Rport to put a floating Dollar sign in a numeric field when it is converted to ascii. The dollar command must be specified after the input file has been specified. Currently Floating is the only option for a given field.

Exit Command [E]

Exit RPort in one of three ways. By default, perform the current task, if any, then leave RPort. Users are often frustrated when they exit RPort after specifying part of a task and RPort starts processing the task. To avoid this situation, use the Abort or Suspend options to exit RPort conveniently without executing the current task.

EXIT [ABORT | SUSPEND | XEQ]

(Default: Xeq)

Typing Exit with no parameters means Exit Xeq. RPort recognizes special command names which specify both the Exit command and an exit option (e.g., ES means Exit Suspend).

Exit Abort [EA]

Cancels the current operation and terminates RPort. The Exit command without parameters always attempts to perform the task currently specified, while Exit Abort cancels the task and terminates immediately. Should RPort be executed as a son process, Exit only suspends RPort, while Exit Abort actually terminates the process.

Examples

```
%:comment.  You began to specify an input file, stopped for
%:comment.  coffee, and decided to cancel the task
%:comment.  upon your return.
%input invoices
... coffee break ...
%exit abort                                {cancel the task and terminate}
End Of Program
```

Exit Suspend [ES]

When running RPort as a son process (e.g., from Qedit), it would be nice to suspend RPort without executing the current task. Exit Suspend does this. After returning to RPort, all specifications for the current task are still in effect.

Examples

```
:run qedit.pub.robelle
/:run rport.pub.robelle
%input invoices
%floating fixed 2                          {start specifying options}
%exit suspend                              {return to Qedit without an Xeq}
STEXPORT.PUB.ROBELLE is still alive. Okay to HOLD ? Y
/...
/:activate RPort
STEXPORT.PUB.ROBELLE
%output invdata                            {continue specifications}
%xeq                                       {execute the current task}
```

Exit Xeq [EX]

To perform the current task, you can either use Xeq (which leaves you inside RPort, ready to define another task) or Exit Xeq (which leaves RPort when done with the task).

After the RPort task completes, RPort either terminates, or suspends and awakens a father process (i.e., :Run from within Qedit). Exit Xeq is the default option (i.e., specifying exit starts execution of the current task).

Examples

```
:run rport.pub.robelle
%exit                                   {no input was specified}

:run rport.pub.robelle
%input invoices
%floating fixed 2
%output invdata
%exit                                   {format and stop}
```

Floating Command [FL]

Specify the format and the number of decimal-places for floating-point fields.

FLOATING DEFAULT |
FIXED *decimal-places* |
SCIENTIFIC *decimal-places*

(Default: Default)

By default, RPort formats floating-point fields (classic floating-point or IEEE floating-point) into either a fixed number or into scientific notation. Which notation RPort chooses, depends on the value of each field in each input record. You can force RPort to choose either scientific or fixed notation and the number of decimal places for all floating-point numbers. You cannot specify these options for a specific field or make them different for 32-bit versus 64-bit floating-point numbers.

Fixed Format

Use Floating Fixed to force all floating-point numbers to appear in a fixed format. You specify the maximum number of digits to the right of the decimal point. If you specify Floating Fixed, RPort does not remove trailing zeros from the formatted numbers. If you specify Columns Fixed, all floating-point values will be aligned along the decimal point.

Scientific Format

Scientific Format is currently unavailable in RPort at this time.

Use Floating Scientific to force all floating-point numbers to appear in scientific notation. You must specify the number of digits to the right of the decimal point. The Scientific option formats the number with all significant digits to the right of the decimal-point followed by the exponent (e.g., "0.47832E-10").

Notes

Both the Fixed and Scientific options attempt to round the number to the specified number of decimal-places within the maximum width for each floating-point data-type. If RPort cannot format a floating-point field in the specified number of decimal-places, the number appears as asterisks "****".

Form Command [F]

Display the fields in a self-describing file.

FORM [*filename*]

If no file name is specified, the fields in the input file are displayed. The display shows the field type and field length in IMAGE notation. An I1-field is a single integer. Packed-fields show the number of nibbles (subtract one to obtain the number of digits). Byte and zoned-decimal fields show the byte length.

When showing the form of a self-describing file, RPort shows the byte offset of each field after the subcount, type, and sublength. The first field always appears at offset one.

There are two types of self-describing files. One type is produced with Suprtool's Query output option. You produce the other type with the Link output option. The Form command shows the internal self-describing version number, enabling you to tell the difference.

A.00.00 - Query Output Option

Compound fields have a question mark for the type, and the length is the number of bytes in the field. Sort information about the file is missing. Here is an example form listing:

```
%form custfile
File: CUSTFILE.EXAMPLE.ROBELLE   (SD Version A.00.00)
   Entry:                Offset
      CHARACTER          X5    1         {length is five bytes}
      ZONED              Z5    6         {room for five digits}
      INTEGER            I1   11         {single integer}
      DOUBLE             I2   13         {double integer}
      PACKED             P6   17         {room for five digits}
      QUAD               I4   20         {eight-byte integer}
      REPEATINT          ?6   28         {compound field}
      LOGICAL            K1   34         {single logical}
      DBLLOG             K2   36         {double logical}
Limit: 10000  EOF: 15  Entry Length: 44  Blocking: 64
```

B.00.00 - Link Output Option

These self-describing files contain information about how the file was sorted. Compound fields are handled correctly, so the Form command shows compound fields just as you would see them in IMAGE. The Item command in Suprtool identifies the date format or the number of decimal places of an item. The Link output option saves the date and decimal attributes as part of the field description:

```
%form custfile
File: DATAFILE.EXAMPLE.ROBELLE   (SD Version B.00.00)
   Entry:                Offset
      CHARACTER          X5    1  <<Sort #1 >>
      REPEATINT         3I1    6        {compound field}
      DATE               J2   12  <<YYYYMMDD>>
      DOLLAR             P6   16  << .2 >>
Limit: 10000  EOF: 15  Entry Length: 16  Blocking: 64
```

Formout File

The Form command writes all output to the file Formout. This file defaults to %stdlist. You can redirect this file to a line printer or a disc drive. If you redirect the Formout file to a disc file, Suprtool assumes a temporary file by default.

```
>:file formout;dev=lp
>form custfile                          {writes to line printer}
>:file formout;dev=disc
>form invfile                          {writes to temporary file}
```

Heading Command [HEA]

Specify a heading, if any, that appears as the first record of the output file.

HEADING <*number*> NONE | FIELDNAMES | ADD *string* | COLUMN *string*

(Default: None)

Use the Heading command to specify what RPort should write to be aligned with the fields in the Report. You can specify up to four lines of Headings using the number parameter.

Field Names

If you specify Heading Fieldnames, RPort creates a default heading. This heading is constructed by using the field name of each field in the input file. The Fieldname option uses the formatting options that apply to byte-type fields to determine the final format. Each verb for each column of the report is drawn from the sd information, specifically the fieldname. If the fieldname is CHAR-FIELD, the Heading would be constructed as CHAR, for line 1 and FIELD for line 2. Rport will do this for up to four lines of heading. The separator characters are as follows:

```
+ * / ` # % & _ and -
```

RPort produces multiple field names for compound fields. For compound fields, the repeat count is used to determine the number of field names. The repeat count is appended to the field name, starting with one, until all the field names have been generated.

Add String

Because the maximum length of an RPort input line is 256 characters, you may not be able to specify a long heading with a single Heading command.

If you specify your own heading, RPort does not attempt to apply any formatting options. If you need each field in the heading line to be surrounded by quotes and separated by commas, you have to supply these yourself.

For example:

```
head 1 Add "    MyByte MyInt      MyDbl              MyQuad
MyDisp"
head 1 Add "                      MyPack       IEEE4
Ieee8"
head 2 add "     Field Field      Field               Field
Field"
head 2 add "                       Field       Field
Field"
```

Column Headings

You can specify a Heading to align within a column by specifying the Column keyword and the Heading will be aligned with each field from left to right.

```
Heading 1 Column 'Account'
Heading 1 Column 'First Name'
Heading 1 Column 'Last Name'
Heading 1 Column 'City'
Heading 1 Column 'State'
```

Notes

You cannot combine the Add and Column options. You must specify one or the other. If you start with Heading Add and then later specify Heading Column, RPort erases the heading you created with Heading Add and starts over with the first column that you specify with Heading Column. Similarly, if you start with Heading Column, a Heading *string* or Heading Add will start over with a new heading.

Help Command [H]

Show what commands and options are available in RPort.

HELP [*command* | *keyword* [,*section*]]

(Default: browse through the entire help file)

Command Help

If you specify any parameters, Help first assumes that you want help on a specific RPort command. If you know the structure of the help file, you can specify one of the keywords under the command name.

```
%help sign                          {help on the Sign command}
%help sign,trailing                 {trailing section of the Sign command}
```

Keyword Help

If we cannot find any help in the "Commands" section of the help file, we assume that you specified one of the outer-level keywords in the help file. To see this list of keywords, type help with no parameters. You see a short introduction to RPort and then a list of keywords. You can specify any of these keywords on the Help command. You can also specify a subkeyword.

```
%help intro,input                   {input section of Introduction}
```

Quick Help - HQ

HQ asks RPort to look under the keyword Quick in the help file. Quick contains the text from the RPort Quick Reference Guide, offering the experienced user a quick review of the syntax of any command.

```
%hq input                           {quick description of Input}
```

Notes

If no parameters are specified, Help allows you to browse through the help file, rport.Help.Robelle. The Help command uses the Qhelp subsystem from the QLIB. For "help in help", type "?" when you see the Qhelp prompt character ("?"). The help file is organized into levels. To go back to the previous level, press Return. Press F8 to exit the Qhelp subsystem and return to RPort.

Hide Command [HI]

The Hide command tells Rport to not list a field in the report.

Hide [*fieldname* [,*fieldname*]]

(Default: None)

Command Hide

This is especially useful for fields that may not be printable, such as fields that have been re-defined to something other than their native type.

You must specify the Hide command after the input file so Rport can find the fieldname specified. You can specify a series of fieldnames separated by commas.

```
%hide acctsfx
```

Input Command [I]

There can be only one Input file per task. The Input file must be a self-describing file, which should be created by Suprtool using the Output-Query or Output-Link option. If you want RPort to format date-fields and implied decimal places, you must use the Output-Link option of Suprtool when you create the file for input to RPort.

Every record in the input file is formatted into a corresponding record in the output file. It is best to have Suprtool Extract only the fields you actually need.

Listredo Command [LISTREDO]

The Listredo command displays any of the previous 1000 commands.

LISTREDO [*start* [/ *stop*]] [;ABS] [;OUT=*file*]
[*string*] [;REL]
[ALL | @] [;UNN]

(Default: display previous 20 commands)

(BJ and ,, are short for LISTREDO)

Commands are numbered sequentially from one as entered; the last 1000 are retained. You can display a single command, a range of commands, all 1000, or all the commands whose name matches the string. You can print the commands with ABSolute line numbers (the default), RELative line numbers (-5/-4), or UNNumbered. You can write the commands to your terminal or OUT to a temporary file. If you want to redo any of these commands, see Do, Redo, and Before.

Examples

```
%listredo 5
%listredo 5/10
%listredo help                          {print all Help commands}
%listredo -10                           {print last ten commands}
%listredo ALL                           {print entire redo stack}
%listredo @;rel                         {print ALL, relative numbers}
%listredo purge                         {print all Purge commands}
%listredo purge xx                      {print all "purge xx" commands}
%listredo @purge                        {print all with "purge" anywhere}
%listredo 1/10;out=*lp                  {dump commands to printer}
%listredo @;unn;out=save                {write commands to a file}
```

Notes

The Listredo command cannot be abbreviated, but BJ and ,, (comma comma) are accepted as a short forms.

Persistent Redo

Redo commands can be saved in a permanent file and can therefore be used from another session. You can use the **Set redo** command to specify a filename to save your redo commands. Please see the Set Redo command for details.

Output Command [O]

Specifies the Output file.

OUTPUT * | *filename* [TEMP] [ERASE]

By default, the Output file is permanent and named "Output". If you specify Columns None, the output file will have variable-length records. When Columns Fixed is specified, RPort creates the output file with fixed-length records.

RPort computes the maximum record size by computing the maximum length of each formatted field and adding them together. If the heading line is longer than this, the length of the heading line is used as the record length. You can use a :file equation to reduce the record size, but it is not recommended since too short a record size can cause RPort to fail in the middle of a task.

File Equations

RPort allows the use of file equations to help ensure a file is built with the attributes that you require in the output file.

One example of this is when you are generating HTML output and want to use it with the Apache Web server on MPE. The Web server needs the files to be bytestream files.

One caveat for this technique with bytestream files is that you have to specify a disc value greater than or equal to the total number of bytes that will be written to the file, because by definition a bytestream file has a record size of only one byte. An example of this technique is as follows:

```
:file x=myhtml;rec=,,b;disc=100000
%in mysdfile
%html table title "My Web Page"
%out *x
%exit
```

Stdlist

If the output file name is *, each output record is written to stdlist. This is useful for trying out different formatting combinations until you find the one that best fits the application that you want to import data into. For example,

```
%input     sdfile
%output    *
%xeq
%floating  fixed 2                    {change one option}
%input     sdfile
%output    *
%xeq                                  {view the result}
%sign      none                       {change a different option}
%input     sdfile
%output    *
%xeq                                  {and so on}
```

Redo Command [REDO]

Enables you to modify and repeat any of the previous 1000 command lines.

REDO [*start* [/ *stop*]]
[*string*]
[ALL | @]

(Default: redo the previous command)

The Redo command allows you to modify the commands before it executes them. If you do not need to change them, use the Do command. Commands are numbered sequentially from one as entered; the last 1000 are retained. Use the :Listredo command to display the previous commands. You can Redo a single command, a range of commands, or the most recent command whose name matches a string.

The Redo command uses MPE-style editing logic (D, I, R, U and >). The default mode is to replace characters. To delete, type DDDD under the characters to be removed. To insert, type I under the insertion spot, then the new characters. To undo your changes, type U. To append to the end of the line, use >xxx. To delete from the end of the line, use >DD. To replace at the end of the line, use >Rxxx. And to erase the rest of the line, use D>. If you prefer Qedit-style editing (Control-D, etc.), use the Before command instead of the Redo command.

Examples

```
%listf @.soruce                          {'source' is not spelled right}
NON-EXISTENT GROUP.  (CIERR 908)
%redo                                    {redo most recent command}
listf @.soruce                           {last command is printed}
         our                             {you enter changes to it}
listf @.source                           {edited command is shown}
                                         {you press <return> }

%listredo all
%redo 5                                  {redo 5th command in stack}
%redo                                    {redo previous command}
%redo -2                                 {redo command before previous}
%redo 8/10                               {redo 8th through 10th}
%redo -10/                               {redo -10 through last}
%redo purge                              {redo last Purge command}
%redo purge temp                         {redo last "purge temp"}
%redo @temp                              {redo last containing "temp"}
```

Notes

The Redo command cannot be abbreviated.

To save more commands, use a :File command on the file Stexredo before you run RPort:

```
file stexredo;disc=5000
run rport.pub.robelle
```

Reset Command [R]

Cancel the current task.

RESET

Reset closes the current Input file, then resets the Output file name to "Output". Formatting options are not reset, only the task-related commands are reset. If you try to reset an individual command, RPort prints a warning.

Set Command [S]

Enables or disables certain operating options within RPort. These options are not reset by Xeq or Reset commands.

```
SET     [MAPPED          ON|OFF]
        [MAXLEN          ON|OFF]
        [REDO  filename]
        [STATISTICS      ON|OFF]
        [VARSUB          ON|OFF]
        [VARSUBDEBUG ON|OFF]
        [WARNINGS   ]  ON|OFF
         [ZONEDFIX] ON|OFF
```

Mapped

SET MAPPED ON | OFF

(Initially: OFF)

Mapped forces RPort/iX to read the input file using mapped file access. Specifying this option is an error in RPort/V. If the input is not in memory, the wall time performance is worse with Set Mapped On, but CPU time performance is better. You must Set Mapped On before specifying the input file.

Maxlen

SET MAXLEN ON | OFF

(Initially: ON)

Rport will by default calculate the maximum size of a field, including commas, decimals and dollar signs. The previous version just took into account decimals. This is on by default. You can usin the old calculation by adding set maxlen off to a specific task or globally by adding set maxlen off to your rportmgr file.

Redo

SET REDO *filename*

(Initially: unnamed temporary file)

Commands entered at the RPort prompt are saved in something called the redo stack. You can recall commands from the redo stack by using other commands such as Before, Do and Redo. By default, the redo stack is stored in a temporary file and discarded as soon as you exit. This temporary stack is not preserved across RPort invocations.

The new Set Redo command assigns a permanent file as the redo stack, allowing the stack to become available for future RPort invocations. For example, to assign the Myredo file as a persistent redo stack, enter

```
%Set Redo Myredo
```

If the file does not exist, RPort creates it. Otherwise, RPort uses the existing file. All subsequent commands are written to the persistent redo stack. The setting is valid for the duration of the RPort session. As soon as you exit RPort, the setting is discarded. Next time you run RPort, you will get the temporary stack.

If the file name is not qualified, the redo stack is created in the logon group and account. This may be desirable if you want to have separate stacks. If you want to always use the same persistent stacks, you should qualify the name.

The Verify command shows which stack is currently in use. If it shows <temporary>, then RPort is using the default stack. Anything else is the name of the file used on the Set Redo command.

Concurrency

When RPort uses the default temporary stack, it is only accessible to that particular instance of RPort. You can run as many RPort instances as you need and each one gets its own redo stack. With temporary stacks you will never get into concurrency problems.

If you start using a persistent redo stack, however, you might start running into concurrency problems. A persistent redo stack can only be used by one RPort instance at a time. If you try to use a persistent redo stack that is already in use, you will get the following message:

```
%Set Redo Myredo
EXCLUSIVE VIOLATION: FILE BEING ACCESSED  (FSERR 90)
Unable to open file for REDO stack
```

In this situation, RPort continues to use the redo stack active at the time and lets you continue working as normal.

Qedit can also have permanent redo stacks. To prevent products from writing to each other's redo stack, it is advisable to have separate stacks for each product by giving them different file names. For example, if you use the command

```
set redo myredo
```

you will have a redo stack called Myredo for your RPort commands. If you exit RPort, then run Qedit and supply the same command Set Redo command, your Qedit commands will be written to the same file that was used for your RPort commands.

Statistics

SET STATISTICS ON | OFF

(Initially: OFF)

Statistics causes RPort to print statistics at the end of each task.

Varsub

SET VARSUB ON | OFF

(Initially: OFF)

Setting Variable Substitution causes RPort to resolve any CI variables in a command before processing.

VarsubDebug

SET VARSUBDEBUG ON | OFF

(Initially: OFF)

Suprtool, now has a setting called Set VarsubDebug on which will print out the line after the variable substitution has occurred. This setting only works if Set Varsub is on and Set VarsubDebug is on.

```
setvar outfile &
:"/GREEN/SUPRTEST/filename901234567890123456789012345678901234567890123
45678901"
:run suprtool.pub.robelle
SUPRTOOL/iX/Copyright Robelle Solutions Technology Inc. 1981-2007.
(Version 6.2 Internal)  TUE, OCT 30, 2007,  2:58 PM  Type H for he
>set varsub on
>set varsubdebug on
>in file1sd.suprtest
vd:in file1sd.suprtest
>output !outfile,link,temp
vd:output /GREEN/SUPRTEST/filename90123456789012345678901234567890123
vd:2345678901,link,temp
```

The output is formatted into 74 byte chunks and printed with a preceding “vd:” so the “substituted” line is clear.

Warnings

SET WARNINGS OFF

(Initially: ON)

Suprtool normally prints warning messages out to %stdlist. You can turn off these messages when you are running from batch by issuing a Set Warnings off command. If you are simulating batch mode with the Set Interactive Off command, you must do the Set Warnings off after the Set Interactive Off.

The default for this setting is On.

ZonedFix

SET ZONEDFIX OFF

(Initially: OFF)

Set ZonedFix has been added to fix a zoned field that when converted to byte, results in "?", meaning that the source field has some characters that were not expected in a Zoned field. If Set ZonedFix is On RPort zeroes out the field after an attempted conversion of the field yields characters that cannot be converted.

The default for this setting is Off.

Sign Command [SI]

Specify what should be done with the sign character for numeric-fields.

SIGN NONE | FLOATING | LEADING | TRAILING

(Default: Floating)

All numeric-type fields, except logical fields, have a sign. Integer and floating-point fields can have either a space " " (for positive values) or a negative sign "-". Packed- and display-type fields can have a space " " (neutral), a plus sign "+" (for positive values), or a negative sign "-".

Specify Sign None to cause RPort to completely ignore the sign. If you specify Sign None, no error or warning message appears if any numeric-types have a negative value.

Leading vs. Floating

If you specify Columns Fixed, it is easy to see the difference between a leading versus a floating sign. A leading sign always appears in the same column whereas a floating sign always appears before the first digit of a number. For example,

```
Sign Leading            Sign Floating
-   22415               -22415
-     207                 -207
-   16600               -16600
-   21910               -21910
-    8411                -8411
-      42                  -42
-   16713               -16713
-    7970                -7970
```

Trailing Sign

Specify Sign Trailing to cause the sign character to appear after each formatted number. Remember that for many numeric-types the sign for positive numbers is a space. RPort always leaves room for the sign, even if it is a space.

Size Command [SIZe]

The Size command specifies the total size of the field when converted from numeric to ascii.

SIZE *fieldname number*

(Default: Depends on Data Type)

All numeric-type fields are converted to Ascii using the maximum size that a field can be when converted to ASCII.

Field Format	Output Size
I1, J1	6 bytes
I2, J2	11 bytes
I3, J3	16 bytes
I4, J4	20 bytes
K1	5 bytes
K2	10 bytes
E2	12 bytes
E4	23 bytes
R2	12 bytes
R4	23 bytes
Z*n*	*n*+1 bytes
P*n*	*n* bytes

The Size command lets the user override this maximum size. For example, you could have a field that is a double integer and while the maximum size of the field may be 11 digits, the data may only be no greater than 6 digits. You could then specify:

```
Size dbl-field 7
```

Which would stuff the result of the conversion of the dbl-field into a 7 character space instead of the default 11, and the report will look more reasonable given the data to type disparity.

Subtotal Command [SU]

Subtotal a numeric-field based on a particular sort break.

Subtotal *fieldnames*

(Default: No Subtotals)

The Subtotal command allows a user to specify what numeric fields you want to Subtotal. The subtotal control break is based on the sort order of the self-describing field. The formatting of the Subtotal will use the rules associated with the column that is being subtotaled, including the Size command.

You can specify a single field, or a list of fields separated by commas or spaces.

```
in somefile
subtotal int-field dbl-field
out report
```

Title Command [Ti]

The Title command allows a user to specify up to Title Lines for a given report.

Title Line 1 | 2

LEFT | LEFTMID | CENTER | RIGHT | RIGHTMID

RPNAME | RPDATE * | RPCOMPANY | RPTITLE

RPRUNTIME * | RPPAGENO* | RPGENERIC1 | RPGENERIC2

(Default: None)

Lines and Areas

The Title command breaks up two possible Title lines and into five distinct Areas.

The Areas are Left, LeftMid, Center, RightMid and Right. The Left side of the report, starts and column 1, and the Right Side of the report is at the far right side of the calculated width of the report. The Center area is in between the left and right margins.

Leftmid and RightMid are in the middle of the Left and Center and Center and Right Areas respectively. It is advisable to only utilize the two MID areas on reports with more fields to allow for more space if you are using all or most of the areas.

Components

There are eight components that you can specify in the five different areas across the two lines.

RPDate, RPRunTime and RPPageNo all allow for a TAG to be specified. So, when you specify the Component, RPPageNo, you can have a Tag to be placed to the left of the Component, denoting what that Component is. When you specify:

```
%title line 1 right RpPageNo Tag "Page No: "
```

The execution of the Title command will place components in the areas from left to right and if an overlapping will occur in the last area will overwrite the previous component.

A sample set of Titles could be:

```
%Title Line 1 Left RPDate Tag "Date :"
%Title Line 1 Center RPTitle "Over Draft Report"
%Title Line 1 Right RPPageNo Tag "Page No :"
%Title Line 2 Left RPName "OverDraft"
%Title Line 2 Center RPCompany "Robelle Bank"
%Title Line 2 Right RPRunTime Tag "Run Time: "
```

When the Report is being run, the Titles are the first things written to the Output file, followed by the Headings and then the Data Lines.

Total Command [To]

The Total command for numeric-fields will print a Total of the field specified.

Total *Fieldname(s)*

(Default: None)

The Total command allows you to specify what fields you want to have totaled and on the report. The Total command allows for multiple fieldnames to be specified on one line and can be separated by spaces and or columns.

```
in somefile
total int-field dbl-field
out reportfile
```

Use Command [U]

Specifies a file of commands to be executed as a group.

USE[Q] *filename*

Examples

A usefile makes your task easier by allowing common commands to be specified once in an external file. For example, the following usefile contains all the commands for creating the Invcust file:

```
%use invuse
input    invoices                    {input file to format}
floating fixed 2                     {formatting option}
output   invdata                     {produces the file we want}
exit
```

RPort prints the lines in the usefile, including the comment lines. This allows you to include instructions and reminders in the usefile. In the example above, there were no commands for the user to enter.

Notes

Usefiles cannot be nested in RPort. The usefile may be any unnumbered text file or a Qedit workfile, but no more than 256 characters per record are processed.

By default, RPort displays the commands in a usefile as they are executed. RPort can execute commands *quietly* using the Useq command. For compatibility with Qedit, Useq can be abbreviated to UQ.

Verify Command [V]

Print the definition of the current task.

VERIFY

Verify prints the current Input and Output files and all export specifications; in other words, it is a Verify All command.

Xeq Command [X]

Perform the current task.

XEQ

Xeq checks that you have specified an Input file and an Output file. Then it performs the task and creates the Output file. Finally, it closes the files, ready for you to specify another task or Exit. If you also wish to leave RPort after completing the task, use Exit instead of Xeq.

Zero Command [Z]

Specify whether leading zeros are to appear in numeric fields.

ZERO NONE | LEADING

(Default: None)

Use Zero None to force all numeric fields to have leading zeros removed. If a numeric field has implied decimal places, RPort always formats the number with at least one digit to the left of the decimal place, even if it is zero.

Use Zero Leading to force all numeric fields to be zero-filled. In this case, Sign Leading and Sign Floating both cause the sign to appear in the same place (in front of the leading zeros).

Example of RPort Output

Example

In this example we show you how to use Suprtool and RPort. We start with an IMAGE/SQL dataset, identify the fields that are dates and the number of implied decimal places in other fields. We then produce a self-describing file using the dataset as input and a sample simple report from Rport.

The Form command displays the fields in a dataset or a self-describing file. For files, this information is stored in the user labels of an MPE file and is not accessible with other tools. Use the Form command to obtain the record layout of RPort input files.

Sales File

We will be formatting data from an IMAGE/SQL sales dataset that has the following form:

```
>form d-sales
Database: STORE.DEMO.ROBELLE

D-SALES          Detail    Set    5
    Entry:                    Offset
       CUST-ACCOUNT         Z8      1  (!M-CUSTOMER)
       DELIV-DATE           J2      9
       PRODUCT-NO           Z8     13  (M-PRODUCT)
       PRODUCT-PRICE        J2     21
       PURCH-DATE           J2     25
       SALES-QTY            J1     29
       SALES-TAX            J2     31
       SALES-TOTAL          J2     35
    Capacity: 602 (14)  Entries: 8  Highwater: 8  Bytes: 38
```

Dates and Decimal Places

We use Suprtool's Item command to identify which of the fields in the d-sales dataset are dates and which fields have implied decimal places:

```
>item deliv-date    ,date    ,yyyymmdd
>item purch-date    ,date    ,yyyymmdd
>item product-price ,decimal ,2
>item sales-tax     ,decimal ,2
>item sales-total   ,decimal ,2
```

Salefile

We now produce a file called "salefile" using Suprtool's Output,Link option. The Link option produces a self-describing file, complete with the date and decimal-place information:

```
>get d-sales
>item deliv-date,date,ccyymmdd
>item purch-date,date,ccyymmdd
>item product-price,dec,2
>item sales-tax,dec,2
>item sales-total,dec,2
>out salefile,link
>xeq
>form salefile

File: salefile           (SD Version B.00.00)
  Entry:                    Offset
     CUST-ACCOUNT          Z8      1
     DELIV-DATE            I2      9     <<YYYYMMDD>>
     PRODUCT-NO            Z8     13
     PRODUCT-PRICE         I2     21     << .2  >>
     PURCH-DATE            I2     25     <<YYYYMMDD>>
     SALES-QTY             I1     29
     SALES-TAX             I2     31     << .2  >>
     SALES-TOTAL           I2     35     << .2  >>
Limit: 108  EOF: 8  Entry Length: 38  Blocking: 107
```

Notice how the Form command correctly identifies which fields are dates and which fields have implied decimal places. RPort uses this information to format the file.

RPort Output

We then use RPort to read the self-describing Salefile to produce our sample output on stdlist. To demonstrate how dates are handled, we insert a separator in each date field:

```
:run rport.pub.robelle
in salefile
title line 1 left RPNAME "NeilReport"
title line 1 center RPCOMPANY "My Company Inc."
title line 1 right rppageno tag "Page No : "
title line 2 left rpdate tag "Date : "
title line 2 center rptitle "SalesFile Report"
title line 2 right rpruntime tag "Run Time : "
Heading 1 col "Customer"
Heading 2 col "Account"
Heading 1 col "Delivery"
Heading 2 col "Date"
Heading 1 col "Product"
Heading 2 col "Number"
Heading 1 col "Product"
Heading 2 col "Price"
Heading 1 col "Purchase"
Heading 2 col "Date"
Heading 1 col "Sales"
Heading 2 col "Qty"
Heading 1 col "Sales"
Heading 2 col "Tax"
Heading 1 col "Sales"
Heading 2 col "Total"
date ccyymmdd "/"
size cust-account 9
size sales-total 9
size sales-tax 9
Total sales-tax
total sales-total
out *
xeq

NeilReport             My Company Inc.                      Page No : 1
Date : 20191211       SalesFile Report                  Run Time : 14:05
Customer Delivery Product    Product Purchase Sales    Sales      Sales
Account      Date  Number    Price    Date    Qty       Tax      Total
 10020 1997/10/04 50511501    98.31 **********   2     27.53     224.15
 10003 1997/10/16 50511501    98.31 1997/10/16   1     13.76     112.07
 10003 1997/10/16 50512501   145.62 1997/10/16   1     20.39     166.00
 10003 1997/10/16 50513001   192.20 1997/10/16   1     26.91     219.10
 10016 1997/10/20 50521001    24.59 1997/10/20   3     10.33      84.11
 10016 1997/10/20 50532001   139.85 1997/10/20   1     19.58     159.42
 10020 1997/10/28 50512501   146.60 1997/10/28   1     20.52     167.13
 10010 1997/10/20 50533001    69.92 1997/10/20   1      9.79      79.70
                                                        148.81    1211.68
```

There are no byte-type fields in the input file, so all fields are converted from their internal numeric representation to a string of digits. All date fields were converted from their internal yyyymmdd format to the external yyyymmdd format with a dash separator between day, month, and year. All fields with implied decimal places have been converted with a decimal point.

Limits Within RPort

Maximums

To achieve maximum speed, RPort/V stores all the necessary information in the HP e3000 stack. This limits the size of input and output records and restricts the number of fields per file. Users should be aware of these limits before attempting to use RPort with extremely complicated databases. RPort/iX stores information in a simulated HP e3000 stack, so all the limits apply to both RPort/V and RPort/iX.

Input File - Maximum Record Size - 8192 Bytes

We recommend that you use Suprtool's Extract command to minimize the input record size.

Input File - Maximum Block Size - 8192 Bytes

By default, Suprtool restricts the maximum block size to 4,096 bytes. You can use the Set Blocksize command to increase this size up to 16,384 bytes (8192 words). If you increase the maximum blocksize, it is likely that Suprtool will produce an output file that RPort cannot read.

Input File - Maximum Fields - 255

If you must have more than 255 fields, use Suprtool's Define and Extract commands to extract several fields as one contiguous series of bytes.

Output File - Maximum Record Size - 8192 Bytes

When formatting many fields, it is possible to produce large output records. Once again, using the Extract command to minimize the size of the input records will avoid large output records.

The total length of the Heading line in the output file is also restricted to 4096 bytes.

Welcome to Dbedit

Introduction

Welcome to version 6.2 of Dbedit for MPE, a module of Suprtool that permits people to add, change, list, or delete individual records or "chains" of records from an IMAGE/3000 database. Dbedit is useful for debugging applications, for quickly prototyping systems, and for the data entry of simple applications.

The functions of Dbedit are similar to QUERY, but the commands and operations are more consistent and logical. Because Dbedit is a part of Suprtool, you can hold Suprtool as a suspended process from within other software (e.g., Qedit) with the database open. This facilitates fast process switching when you need to examine a test database.

You enter Dbedit via the Edit command of Suprtool. Once in Dbedit, you cannot use the Suprtool commands (while in Suprtool you cannot use the Dbedit commands). Certain commands are the same in both Dbedit and Suprtool (e.g., Use, Before, and Set). The Before command works independently and each software module saves its own last command.

Restrictions

1. Most Dbedit commands require you to have opened the database using the Base command of Suprtool. Dbedit does not have a Base command.
2. Dbedit does not work with any files other than IMAGE/3000 datasets.
3. You cannot switch to another database while in Dbedit. Instead, you must Exit, do a Base command in Suprtool, then Edit.
4. The maximum size of any individual data item is 80 bytes (i.e., 5X80 is acceptable, but X100 is not).
5. Only datasets whose search fields are compatible with Dbedit can be accessed (i.e., no K5 search fields).

Functions of Dbedit

There are five major functions in Dbedit:

Command	Function
ADD	Add new entries to a dataset.
CHANGE	Change a master search value in all related datasets.
DELETE	Delete entries from a dataset.
LIST	List the value of entries in a dataset.
MODIFY	Modify specific fields of an entry from a dataset.

Performance of Dbedit

Suprtool was designed to be as fast as possible, while Dbedit was designed to have as many features as possible. Dbedit does no special optimizations. It uses the standard IMAGE intrinsics to do all of the accesses to the database. Dbedit does not use the fast sequential access method of Suprtool, but Dbedit usually works only with a few records within your database at one time.

Field Lists

Dbedit arranges the list of fields in a dataset different from QUERY or Suprtool. The QUERY ADD command prompts for each field in the dataset in the order they were declared in the IMAGE schema. In Dbedit, the order of field lists is changed using the following algorithm:

1. The search field for a master dataset or the primary search field for a detail dataset appears first.
2. Any other detail search fields appear second.
3. Any sort fields appear third.
4. All other non-search and non-sort fields that are compatible with Dbedit appear last.
5. Where a non-compatible field would appear in a list, Dbedit replaces it with a compatible field from the end of the list.

Example:

The following example shows the difference between QUERY and Dbedit. We add an entry to the d-inventory dataset of the Store database. In this dataset, supplier-name is the primary search field and product-no is another nonprimary search field.

QUERY/3000

```
>add d-inventory
BIN-NO          =>>1201
LAST-SHIP-DATE  =>>
OH-HAND-QTY     =>>
PRODUCT-NO      =>>2001001
SUPPLIER-NAME   =>>STD Ribbons
UNIT-COST       =>>
```

DBEDIT/SUPRTOOL

```
#add d-inventory
  SUPPLIER-NAME >STD Ribbons_____
     PRODUCT-NO >2001001__
         BIN-NO >1201___
 LAST-SHIP-DATE >___________
    ON-HAND-QTY >___________
      UNIT-COST >________
```

Database Locking

Dbedit uses the following locking strategy. The Add command locks one dataset (using DBLOCK, mode-3) after all of the field values have been entered. The Modify and Delete commands do the following:

1. After all of the field values have been entered, the dataset is locked.

2. The records are re-read using DBGET, mode-4 for details or DBGET, mode-7 for masters.
3. The re-read record is compared with the original record. If they are not the same, no update or delete is done.
4. The record is updated or deleted. When a search field or a sort field is changed with the Modify command, the record is deleted and added again.
5. The dataset is unlocked.

The Change command locks the *entire* database while all key values are being changed. In all cases, the DBLOCK is done unconditionally. This means that Dbedit always waits for other locks to be released (possibly holding up your terminal).

Decimal Points

If you use Suprtool's Item command to specify implied decimal places for an item, Dbedit scales all input values according to the number of implied decimal places. When listing records, fields with implied decimal places are formatted with a decimal point. For example,

```
>item unit-cost,decimal,2                {two implied decimal pts.}
>edit
#add d-inventory:unit-cost
   SUPPLIER-NAME >STD Ribbons
      PRODUCT-NO >2001001
       UNIT-COST >10.50

SUPPLIER-NO     = 5051
PRODUCT-NO      = 50513001
UNIT-COST       = 10.50
```

Type ? to get a description of the field.

When Dbedit prompts for field values, you can type a question mark to obtain a description of the field. If the field has implied decimal places, the field description will include the number of decimal places (e.g., << .2 >> for two implied decimal points).

Critical-Item Update

Starting with MPE/iX 4.0, IMAGE was enhanced to have a new feature: critical-item update. This feature allows programs to change critical fields in detail datasets (search fields and sort fields) by using DBUPDATE. Prior to critical-item update, programs had to use DBDELETE and DBPUT to change critical fields.

By default, IMAGE databases have critical-item update disabled. It is enabled using the HP DBUTIL program. Enabling critical-item update allows Dbedit's Change command and Modify;Updatekey to execute much faster. There are two choices in DBUTIL for enabling critical-item update:

1. Set CIUPDATE = On.
2. Set CIUPDATE = Allow.

The first option turns on critical-item update for all programs. The second option allows user programs (like Dbedit) to enable critical-item update as needed. The second option is the safest, since some programs depend on IMAGE giving an error when they attempt to change a critical field.

Dbedit Commands

General Notes

When you run Dbedit, it prompts for commands on Stdlist with a "#" character and reads command lines from Stdin. Dbedit commands have a command name followed by one or more parameters separated by semicolons, colons, and commas. **Semicolons are NOT used to combine several commands on the same line as in Suprtool.**

This chapter describes the Dbedit commands in detail. The commands are presented in alphabetical order. Each command name is followed by the minimal abbreviation for the command, in brackets. For example: [F] for File and [FO] for Form.

Abbreviating

You may shorten command names to any substring that uniquely defines the command. For example, Add can be shortened to AD or A, since there are no other commands that start with "A". Form, however, can be abbreviated only to FO, since there is a File command, abbreviated F, in Dbedit.

```
>base store.pub,5
>edit
#l m-customer;all                    {list}
#e                                   {exit}
```

Uppercase or Lowercase

You may enter letters in either uppercase or lowercase, because Dbedit upshifts everything in the command line. These two commands are identical:

```
#LIST M-CUSTOMER
#list m-customer
```

Continuation

The maximum *physical* command line is 256 characters. You may enter commands on multiple input lines by putting an "&" continuation character at the end of the line. The maximum *total* command length is 256 characters. Multiple commands cannot be placed on one input line. The separating semicolon, colon, or comma in commands is REQUIRED, not optional.

```
:run suprtool.pub.robelle
>base store.pub,5                          {open the database in Suprtool}
>edit                                      {enter Dbedit}
#list m-customer                           {use all of the defaults}
#list m-customer;all                       {list all records in m-customer}
#exit                                      {return to Suprtool}
```

Control-Y

If you press Control-Y during an operation, Dbedit responds by printing a blank line and stopping the current operation.

Comments on Command Lines

Comments may appear at the end of any command line, when they are surrounded by braces. Many of the examples in this manual show comments at the end of each command line. You can enter a comment as the only item in a Dbedit command line. You can also enter comments after field values.

```
#             { List all related records for a specific supplier. }
#list m-supplier;related    {Must include Related keyword}

   SUPPLIER-NAME  >STD Ribbons   {supplier to search for}
```

If you want to include a brace as part of an x-type field, you must precede the brace with the Dbedit escape character "[". For example, to search for the supplier "{STD Ribbons}", you would use:

```
#list m-supplier            {include braces in the name}
   SUPPLIER-NAME  >[{STD Ribbons}
```

MPE Commands

Dbedit interprets any command line beginning with a colon (:) as an MPE command. Only the commands that MPE allows in "break" are allowed in Dbedit. This feature can be used to establish :File commands for the SUPRLIST file, to show the time, and to include :COMMENT lines. For example:

```
#:comment Modify M-CUSTOMER records
#modify m-customer
#exit
```

MPE/iX Commands

Dbedit/iX will execute any MPE command (e.g., Run), UDCs, and command files. **Caution:** programs that suspend, instead of terminating are not killed by the HPCICOMMAND intrinsic.

Calculator

Any command line beginning with an equal sign (=) is treated as a calculator expression. You may use this feature to compute data entry values without the need of an electronic calculator.

```
=2745*1.33
Result= 3650.85
```

Example Database

The examples in this manual use the revised STORE database described in the *IMAGE/3000 Handbook*.

Prompting for Search Criteria

In the Change, Delete, List and Modify commands, Dbedit first prompts you for **search criteria** and then processes the records you have selected. Search criteria are any or all of the search and sort fields of the file. Dbedit asks for the value of the primary search field first, unless you override the prompt ordering with the KEY option. For detail datasets, it then asks for match values for the other search fields and sort fields. You may press the Return key to any of these prompts to indicate that you don't care what values these fields have.

When Dbedit finishes processing the records you select, it recycles and prompts you for the next set of search criteria. You may press the Return key at this point to exit from the command and return to the # prompt.

Command Parameters

The major commands (File, List, Add, Delete, Modify, and Change) have a similar parameter structure, consisting of the command name, then an optional *file* part and an optional *option* part. A space separates the *file* part from the command name, and a semicolon separates the *option* part from the *file* part. The general format of these commands is

#command [*file*] [;*options*]

File Parameter

The file parameter consists of an IMAGE/3000 dataset name followed by an optional list of field names. If the *file* part is missing, Dbedit uses the previous file. The general format of the file parameter is

#command [*file*] [:*fieldname*,...]

Even when you use field names, Dbedit will add the search fields to the field list. In the Add command, Dbedit assumes default values for noncritical fields that are missing, but will prompt for the search fields and sort fields (they are required).

```
#add d-inventory:bin-no                    {assume defaults for all but BIN-NO}
   SUPPLIER-NAME  >STD Ribbons
      PRODUCT-NO  >105391
          BIN-NO  >10
```

In this case you will not be prompted for LAST-SHIP-DATE, ON-HAND-QTY, or UNIT-COST.

In the Modify command, you can specify a set of fields to modify. Dbedit will not prompt you for new values for any other fields. For example:

```
#mod d-inventory:unit-cost                {only modify UNIT-COST}
   SUPPLIER-NAME  >STD Ribbons            {prompt for search value}
      PRODUCT-NO  >105391                 {prompt for another one}

Enter new values (or <return> to leave as is):
   SUPPLIER-NAME  >STD Ribbons            {prints the search value}
      PRODUCT-NO  >105391                 {prints the other one}
       UNIT-COST  >500                    {prints existing value}
                  _____                   {prompts for new value}
```

In this case you will not be prompted for BIN-NO, ON-HAND-QTY, or LAST-SHIP-DATE.

When working on a single dataset, it is only necessary to specify the dataset name in the first command. For example:

```
#list d-inventory
   SUPPLIER-NAME  >STD Ribbons

#list                                     {use the previous file parameter}

   SUPPLIER-NAME  >//
```

Option Parameter

The *file* parameter and the *options* must be separated by a semicolon.

#command [*file*] [;*option*...]

The available *options* are:

numeric-value, All, Key, Limit, Related, UNder, UPdatekey

These options qualify the operation of the File, List, Modify, Change, Delete, and Add commands. Some options only apply to one command. Options can be combined and can be abbreviated. When more than one option is specified, each option must be separated by a semicolon.

```
#list d-inventory;key=product-no;under
       PRODUCT-NO >_____
```

Numeric-Value Option

Commands normally cycle, prompting for new search values or new entries, until you press Return or Control-Y. However, if you specify a *numeric-value* after the semicolon, the command only prompts you *numeric-value* number of times. For example, if you only want to do one List function, you would enter:

```
#list d-inventory;1                       {only prompt for SUPPLIER-NAME once}
```

All Option

The All option works only with the List, Modify, or Delete commands. When All is specified, every record in the specified file is processed sequentially. You can stop the scan by pressing Control-Y.

Key Option

The Key option overrides the primary search field. Dbedit prompts for the primary search field first. Often, this is not the value that you know. You can use the Key option to force Dbedit to prompt you for another search field. For example:

```
#list d-inventory                          {use defaults}
    SUPPLIER-NAME >STD Ribbons
       PRODUCT-NO >_________

#list d-inventory;key=product-no           {use PRODUCT-NO}

       PRODUCT-NO >105391___
    SUPPLIER-NAME >_________________
```

Limit Option

The Limit option controls the number of entries allowed per key value. This option is only useful for the Add command. Dbedit limits the number of entries for the first field in the field list to the Limit value specified. For example,

```
#add d-inventory;limit=2                   {two records per supplier}
#add d-inventory;limit=2;key=product-no
                                           {two records per product}
```

Related Option

The Related option is for finding or deleting related records. It applies only to Delete and List. If you use Related when Listing an entry in a master dataset, Dbedit prints the specified master entry and then prints all detail entries with the same search value in all datasets that are linked to the master dataset by an explicit path. If you use List Related with a detail dataset, Dbedit prints the specified detail entry, followed by the master entry for each search field in the detail.

You can only Delete Related from a master dataset. Dbedit shows the master entry. After confirming the deletion, Dbedit deletes all entries with the same search value in all related datasets. Control-Y can be used to interrupt the deletion process.

Dbedit can only use explicitly defined IMAGE paths to navigate between datasets. User-defined paths are not supported in Dbedit. When listing related records, Dbedit shows all fields in each dataset including those that Dbedit normally doesn't support (e.g., X100). The following example shows how related records are listed, starting with a master dataset and then its related detail datasets.

```
#list m-supplier;related                   {all related records}
     SUPPLIER-NAME >STD Ribbons

        The M-SUPPLIER record is listed here.

        D-INVENTORY records with a SUPPLIER-NAME of
        STD Ribbons are listed here.
```

Under Option

When Dbedit prompts for a value for a field, it prints a series of underlines. These underlines indicate the maximum field width.

These underlines can be useful, but they may not work on all terminals. You can disable underlining by using Set Underline Off. Once underlining is turned off, you may wish to enable it again, but only for one command. The Under option overrides the Set Underline, but only for the current command.

Updatekey Option

The Updatekey option allows the Modify command to change the value of search and sort fields (i.e., critical fields). Normally, Modify does not allow any changes to the search or sort fields. Without the Updatekey option, Modify does a DBUPDATE of the modified record. If CIUPDATE is Disallowed when you specify Updatekey and change a critical field, Modify will DBDELETE the existing record, then DBPUT a new record with the changed values. If CIUPDATE is On or Allowed in the database, Modify can use DBUPDATE to change the critical fields. This is much faster than using DBDELETE and DBPUT.

Subcommands

You may enter a subcommand any time Dbedit prompts for the value of a field. The available subcommands are:

Subcommand	Purpose
//	stops the current command immediately.
\\	same as // (you may also use the Control-Y key).
?	describes the current field, including any implied decimal points.
<	goes back one field to the previous field in the list.
<3	goes back three fields.
<<	returns to the first field in the list.
>	goes forward one field to the next field in the list.
>3	goes forward three fields.
>>	skips the rest of the fields in the list. This is especially useful when Dbedit is prompting you for multiple search and sort fields and you only want to enter the first.
'	uses blanks for the field (useful in batch).
*	uses the last value for this field.
[	forces what follows the [to be a value and not a subcommand (e.g., [*BOB tells Dbedit to use *BOB as the actual value instead of interpreting the * to mean "last value").
@F	search for the field F (e.g., @bin-no). The field can include a subscript (e.g., @street-address(2)).
=	calculator.

Examples

```
#list m-supplier                          {we'll stop immediately}
    SUPPLIER-NAME >//

#list m-supplier                          {we will start again}

    SUPPLIER-NAME >Standard Type
       PRODUCT-NO ><                      {re-enter supplier-name}

    SUPPLIER-NAME >STD Ribbons
       PRODUCT-NO >>>                     {skips the rest}

#list m-supplier

    SUPPLIER-NAME >STD Ribbons
       PRODUCT-NO >[>575                  {">575" is the part-no}

#modify d-inventory

    SUPPLIER-NAME >STD Ribbons
       PRODUCT-NO >105391
           BIN-NO >@unit-cost             {skip to unit-cost}
        UNIT-COST >

#modify m-customer

     CUST-ACCOUNT >4003302
             CITY >@street-address(2)     {go to subfield}
STREET-ADDRESS(2) >
```

Add Command [A]

Adds new entries to a dataset.

ADD [*file*] [;*options*]

Options: *numeric-value*|Key|Limit|UNder

If no field list is entered as part of the *file*, Dbedit will prompt for all of the fields in the *file*. You may use the ">n" or ">>" subcommands to navigate quickly through the field list, but you must enter values for all search and sort fields.

The database password must give you write access to the entire dataset. The Add command will stop after LIMIT= number of entries have been added for any one key value. Dbedit checks each search field value as it is entered. For master datasets the search field value must not exist. For detail datasets the search field value must exist. To add records from a disc file, see the Put command of Suprtool.

Before Command [B]

Repeat any combination of the previous 1000 command lines, with or without editing.

BEFORE [*start* [/ *stop*]]
[*string*]
[ALL | @]

(Default: redo previous line)

(BQ=redo without change)

The Before command allows you to modify the commands before it executes them. If you don't need to change them, use BQ or Do.

The Before command uses Qedit-style Control characters for modifying the commands. The default mode is to replace characters. To delete use Control-D, and to insert use Control-B. If you prefer HP-style modify (D, R, I, and U), use the Redo command instead of Before.

Examples

```
#listf @.soruce                          {'source' is not spelled right}
NON-EXISTENT GROUP.  (CIERR 908)
#Before                                  {redo most recent command}
listf @.soruce                           {last command is printed}
         our                             {you enter changes to it}
listf @.source                           {the edited command is shown}
                                         {you press <return> }

#listredo -10/
#before 5                                {redo 5th command in stack}
#bef 8/10                                {redo 8th through 10th}
#b listf                                 {redo last Listf command}
#b @temp                                 {redo last containing "temp"}
#before -2                               {redo command before previous}
#before -5/-2                            {redo by relative lines}
```

Modify Operators

If you wish to change any characters within the line, the modify operators are the regular Control Codes used in Qedit:

- Any printing characters replace the ones above.
- Control-D plus spaces deletes columns above.
- Control-B puts you into "insert before" mode.
- Control-A starts appending characters at the end of line.
- Control-A, Control-D, plus spaces, deletes from the end.
- Control-T ends Insert Mode, allowing movement to a new column.
- Control-G recovers the original line.
- Control-O specifies "overwrite" mode (needed for spaces).

Change Command [C]

Changes all uses of a specific search field value in all detail datasets related to a master. This command only applies to master datasets.

CHANGE [*file*] [;*options*]

Options: *numeric-value*|Key|Limit|UNder

This command changes the search field value in all related detail datasets. Dbedit can only change values in detail datasets where there is an explicit IMAGE path. It is up to the user to change any user-defined paths.

Once this command has started making changes to the database, it cannot be stopped. Entering Control-Y during the change will have no effect. Dbedit locks the entire database while all changes are taking place. The database password must give you write access to all related datasets that must be changed.

If CIUPDATE is On or Allowed in the database, the Change command executes much faster.

Example

```
#change m-supplier
Change Key Value for File: M-SUPPLIER

Enter Existing Key Value to Find:
   SUPPLIER-NAME       >Standard Ribbons
Enter New Key Value to Replace with:
   SUPPLIER-NAME       >STD Ribbons

SUPPLIER-NAME   = Standard Ribbons
CITY            = San Leandro    STATE-CODE      = CA
STREET-ADDRESS  = 100 Main St.
           (2)
ZIP-CODE        = 94345

OK to change this entry [no]: y
     Begin changes (be patient) ..  ..  end changes!!
```

Delete Command [D]

Removes entries from a dataset.

DELETE [*file*] [;*options*]

Options: *numeric-value*|All|Key|Limit|Related|UNder

If you are deleting entries from a master dataset, all entries from related detail datasets must be removed first. Before any record will actually be deleted, Delete prints the record and asks you whether it is okay to delete it; the default answer is NO.

The field list of the *file* specifies which fields to list before prompting for verification of the deletion. The ALL *option* allows you to review all entries in a detail dataset and remove some or all of them. The database password must give you write access to the entire dataset. ALL does not work on master datasets; use the Suprtool >Delete command instead.

You can delete records related to a master dataset, but not the other way around. Dbedit does not print the detail records that are deleted. Use the List command with the RELATED option before doing the deletion.

Do Command [DO]

The Do command will repeat (without changes) any of the previous 1000 commands.

DO [*start* [/ *stop*]]
[*string*]
[ALL | @]

(Default: repeat the previous command)

Commands are numbered sequentially from 1 as entered and the last 1000 of them are retained. Use the :Listredo command to display the previous commands. You can repeat a single command (`do 5`), a range of commands (`do 5/10`) or the most recent command whose name matches a string (`do list`). If you want to modify the commands before executing them, use Redo or Before.

Examples

```
#listredo
#do                                  {do previous command again}
#do 39                               {do command line 39 again}
#do 5/8                              {do command lines 5 to 8 again}
#do list                             {do most recent List command}
#do show                             {do last starting with "show"}
#do showjob job                      {do last "showjob job" command}
#do @job                             {do last containing "job"}
#do -2                               {do command before previous}
#do -7/-5                            {do by relative line number}
#do 5/                               {do command lines 5 to "last"}
```

Notes

The Do command cannot be abbreviated.

Exit Command [E]

Leaves Dbedit and returns control to Suprtool.

EXIT

File Command [F]

Establishes the current file, field list, and search field.

FILE [*file*] [;*options*]

Options: *numeric-value*|All|Key|Limit|Related|UNder|UPdatekey

If Set Reset is Off, you can use the File command to specify the KEY= for the specified *file*. For example:

```
#set reset off
#file d-inventory;key=product-no
```

will cause all subsequent commands to prompt for the PRODUCT-NO before the SUPPLIER-NAME. Specifying a new *file* or *options* parameter in the Add, Change, Delete, List, or Modify commands overrides and replaces the current *file* and *option* values.

Form Command [FO]

The Form command displays a description of the items and datasets in a database. It is similar in format to the FORM command of QUERY.

FORM [SETS | ITEMS | PATHS

dataset | *data-item* | *filename*]

(Default: fields in current dataset)

If you request information about a specific detail dataset, Suprtool will print the path information in DBSCHEMA format. The path shows the related master dataset and the sort item-name. The capacity is also shown in DBSCHEMA format. The blocking factor appears after the capacity in parentheses. If dynamic dataset expansion (a feature of MPE/iX 5.0) has been enabled, the initial capacity and the increment are shown next. On MPE/iX 5.0 or later, the highwater mark is shown. The record length in bytes appears last.

When showing the form of a dataset, Dbedit shows the byte offset of each field after the subcount, type, and sublength. The first field always appears at offset one. If you have specified a date format or the number of implied decimal points with the Item command, these attributes will appear as part of the form listing.

The QUERY ALL option is not supported.

Example

```
>base store,5
>item last-ship-date,date   ,yymmdd
>item unit-cost     ,decimal,2
>edit                                    {enter Dbedit}
#form d-inventory
D-INVENTORY      Detail
  Entry:
     BIN-NO               J1    1
     LAST-SHIP-DATE       J2    3             <<YYMMDD>>
     ON-HAND-QTY          J2    7
     PRODUCT-NO           Z8   11  (M-PRODUCT)
     SUPPLIER-NAME        X16  19  (!M-SUPPLIER)
     UNIT-COST            P8   35            << .2  >>
Capacity: 112 Entries: 3 Entry Length: 19 Blocking: 14
```

Formout File

The Form command writes all output to the file Formout. This file defaults to $stdlist. You can redirect this file to a line printer or a disc drive. If you redirect the Formout file to a disc file, Suprtool assumes a temporary file by default.

```
>:file formout;dev=lp
>form sets                               {writes to line printer}
>:file formout;dev=disc
>form d-inventory                        {writes to temporary file}
```

Form Keywords

The Form command shows Items, Paths, and Sets before searching for a dataset or file with these names. Use a string (e.g., "sets") to display the form of a dataset or file that matches one of the Form keywords.

```
>form "paths"
```

Help Command [H]

Show what commands and options are available in Dbedit.

HELP [*command* | *keyword* [,*option*]]

(Default: browse through the entire help file)

Command Help

If you specify any parameters, Help first assumes that you want help on a specific Dbedit command. If you know the structure of the help file, you can specify one of the keywords under the command name.

```
#help list                          {Help on the List command}
#help list,notes                    {notes section of the List command}
```

Keyword Help

If no help is found in the Commands section of the help file, we assume you specified one of the outer-level keywords. To see this list of keywords, type help with no parameters. You'll see a short introduction to Dbedit before the list. Specify any of the keywords, or a subkeyword, on the Help command.

```
#help intro                         {introduction section}
```

Quick Help - HQ

HQ asks Dbedit to look under the keyword QUICK in the help file. QUICK contains the text from the Dbedit Quick Reference Guide, offering the experienced user a quick review of the syntax of any command.

```
#hq add                             {quick description of Add}
#hq commands                        {quick list of command names}
```

Notes

If no parameters are specified, Help allows you to browse through the "help" file. The Help command uses the QHELP subsystem to enable you to move through the file Dbedit.Help.Robelle, which contains most of the User Manual. For "help in help", type "?" when you see the QHELP prompt character ("?"). The help file is organized into levels. To go back to the previous level, press Return instead of a key name. If you press F8, you will exit the QHELP subsystem and return to Dbedit.

List Command [L]

Displays entries from a dataset.

LIST [*file*] [;*options*]

Options: *numeric-value*|All|Key|Limit|Related|UNder

The field list of the *file* parameter specifies which fields of the entry to print (default of course is all of them). Search field values are not listed, unless they are included in the field list. The ALL *option* lists all records in the specified *file*. The RELATED *option* prints related records from other datasets as well as the records you select. The listing is printed to $stdlist, unless you use Set LP On to specify Suprlist as the output file.

The Suprtool >List command also displays selected records, with the option of dumping totally in Octal/Char format. Use >List when you suspect that a dataset contains bad data, or to select from a large dataset (>List is faster than #List).

```
#list m-customer
List in File: M-CUSTOMER

    CUST-ACCOUNT       >4003302

CITY            = Los Altos     CREDIT-RATING  = 100000
CUST-ACCOUNT    = 4003302       CUST-STATUS    = 20
NAME-FIRST      = Ralph         NAME-LAST      = Perkins
STATE-CODE      = CA
STREET-ADDRESS  = Room 655
            (2)   Century Plaza Building
ZIP-CODE        = 93002

#list m-supplier;related                {all related records}
List in File: M-SUPPLIER

SUPPLIER-NAME       >Standard Ribbons

CITY            = San Leandro    STATE-CODE      = CA
STREET-ADDRESS  = 100 Main St.
            (2)
SUPPLIER-NAME   = Standard Ribbons
ZIP-CODE        = 94345

Related Records from the File : D-INVENTORY
  Key to path: SUPPLIER-NAME
BIN-NO          = 1201          LAST-SHIP-DATE  = 840501
ON-HAND-QTY     = 296           PRODUCT-NO      = 105391
     SUPPLIER-NAME   = Standard Ribbons
UNIT-COST       = 500
```

Listredo Command [LISTREDO]

The Listredo command will display any of the previous 1000 commands.

LISTREDO [*start* [/ *stop*]] [;ABS] [;OUT=*file*]
[*string*] [;REL]
[ALL | @] [;UNN]

(Default: display previous 20 commands)

(BJ and ,, are short for LISTREDO)

Commands are numbered sequentially from 1 as entered and the last 1000 are retained. You can display a single command, a range of commands, all 1000, or all the commands whose name matches the string. You can print the commands with ABSolute line numbers (the default), RELative line numbers (-5/-4), or UNNumbered. You can write the commands to your terminal or OUT to a temporary file. If you want to redo any of these commands, see Do, Redo, and Before.

Examples

```
#listredo 5
#listredo 5/10
#listredo help                          {print all Help commands}
#listredo -10                           {print last ten commands}
#listredo ALL                           {print entire redo stack}
#listredo purge                         {print all Purge commands}
#listredo purge xx                      {print all "purge xx" commands}
#listredo @purge                        {print all with "purge" anywhere}
#listredo @;rel                         {print ALL, relative numbers}
#listredo 1/10;out=*lp                  {dump commands to printer}
#listredo @;unn;out=save                {write commands to a file}
```

Notes

The Listredo command cannot be abbreviated, but BJ is accepted as a short form.

Modify Command [M]

Changes the values of any or all fields in a dataset entry.

MODIFY [*file*] [;*options*]

Options: *numeric-value*|All|Key|Limit|UNder|UPdatekey

The field list of the *file* parameter specifies which fields to modify. When the UPDATEKEY *option* is specified, search and sort field values may be changed.

The existing value of each field is printed before a new value is accepted. Entering a carriage return preserves the old value. If a new value is entered, it replaces the old value. The ALL *option* allows you to review and modify all of the entries in a dataset in serial order.

If CIUPDATE is On or Allowed in the database, the UPDATEKEY *option* executes much faster.

```
#modify m-customer:credit-rating
Modify in File: M-CUSTOMER

    CUST-ACCOUNT       >4003302

Enter new values (or <return> to leave as is):

    CUST-ACCOUNT       =4003302
   CREDIT-RATING       =1000.00
                        2500.00
CUST-ACCOUNT    = 4003302      CREDIT-RATING   = 2500.00
```

Q Command [Q]

Prints a message on $stdlist.

Q [string]

(Default: print a blank line)

The string of up to 80 characters is printed on $stdlist. The string should not be enclosed in quotes unless you want the message printed in quotes.

You can use Q to include instructions in usefiles. Use :COMMENT in usefiles for a nonprinting comment line.

Redo Command [REDO]

Enables you to modify and repeat any of the previous 1000 command lines.

REDO [*start* [/ *stop*]]
[*string*]
[ALL | @]

(Default: redo the previous command)

The Redo command allows you to modify the commands before it executes them. If you don't need to change them, use the Do command. Commands are numbered sequentially from 1 as entered and the last 1000 are retained. Use the :Listredo command to display the previous commands. You can redo a single command, a range of commands, or the most recent command whose name matches a string. The Redo command cannot be abbreviated.

The Redo command uses MPE-style editing logic (D, I, R, U and >). The default mode is to replace characters. To delete, type DDDD under the characters to be removed. To insert, type I under the insertion spot, then the new characters. To undo your changes, type U. To append to the end of the line, use >xxx. To delete from the end of the line, use >DD. To replace at the end of the line, use >Rxxx. And to erase the rest of the line, use D>. If you prefer Qedit-style editing (Control-D, etc.), use the Before command instead of the Redo command.

Examples

```
#listf @.soruce                          {"source" is not spelled correctly}
NON-EXISTENT GROUP.  (CIERR 908)
#redo                                    {redo most recent command}
listf @.soruce                           {last command is printed}
         our                             {you enter changes to it}
listf @.source                           {edited command is shown}
                                         {you press <return> }

#listredo all
#redo 5                                  {redo 5th command in stack}
#redo                                    {redo previous command}
#redo -2                                 {redo command before previous}
#redo 8/10                               {redo 8th through 10th}
#redo -10/                               {redo -10 through last}
#redo purge                              {redo last Purge command}
#redo @temp                              {redo last containing "temp"}
```

Set Command [S]

Changes certain operating options within Dbedit. Except for LP, these options are saved when you return to Suprtool and restored if you enter Dbedit again.

```
SET     [LP             ON|OFF ]
        [PROMPT         character ]
        [QUIET          ON|OFF ]
        [RESET          ON|OFF ]
        [UNDERLINE      ON|OFF ]
        [VERIFY         ON|OFF ]
```

LP

SET LP ON|OFF

(Initially: OFF)

All output from the List command is normally sent to $stdlist. When you turn Set LP to On, Dbedit opens the file Suprlist which defaults to the line printer. Turning Set LP Off closes the SUPRLIST file and releases it to the spooler. Dbedit automatically closes the Suprlist file when you return to Suprtool.

Prompt

SET PROMPT *char*

(Initially: #)

Prompt tells Dbedit to use a different character for prompting. Any special character can be used as the prompt character. For example:

```
>edit
#set prompt %
%list m-customer
```

Quiet

SET QUIET ON|OFF

(Initially: OFF)

Turning this option On causes Dbedit to reduce the number of helpful messages that are printed and to shorten other messages.

Reset

SET RESET ON|OFF

(Initially: ON)

When you use the File, List, Modify, Change, or Delete commands you may override the default order that Dbedit uses to prompt for search fields (using the KEY= *option*). With Reset On, Dbedit always resets the KEY= *option* to the default. With Reset Off, Dbedit will continue to use the new order of search fields.

Underline

SET UNDERLINE ON|OFF

(Initially: ON)

When Dbedit prompts for field values, it prints a series of underlines to show the field width. Underline Off disables this feature. Users with slow printing terminals will find Dbedit faster with Underline Off. The Under *option* may be used to temporarily enable underlining for a single command.

Verify

SET VERIFY ON|OFF

(Initially: ON)

When you are adding new records to a dataset, Dbedit lists the contents of new records after you have entered all of the field values. Turning Verify Off causes Dbedit to stop listing new records after they are added.

Use Command [U]

Specifies a file of commands to be executed as a group.

USE[Q] *filename*

Examples

A usefile makes your task easier by allowing common commands to be specified once in an external file. Usefiles can also be set up for data entry functions such as adding a new customer into the database (e.g., add one master entry, add one billto entry, add multiple shipto entries, and then list the master entry plus RELATED details). For example, here is a usefile that uses the File command to define a dataset:

```
file d-inventory;key=product-no;5       {key and limit}
q You may use the List, Modify, or Delete
q commands to modify inventory records.  You
q are restricted to at most five inventory
q records per product number.
```

Use the file inside Dbedit:

```
>base store
>edit                                   {enter Dbedit}
#useq dinvent.use                       {specify commands from the usefile}
You may use the List, Modify, or Delete
commands to modify inventory records.  You
are restricted to at most five inventory
records per product number.
#                                       {Dbedit prompts for commands}
```

Quiet Execution

By default, Dbedit displays the commands in the usefile as they are executed. Dbedit can execute commands *quietly* using the Useq command:

```
#useq store.use                         {no commands are listed}
```

Notes

Usefiles cannot be nested in Dbedit. The usefile may be an EDITOR /KEEP file,UNN or a Qedit workfile, but no more than 256 characters per record will be processed. For compatibility with Qedit, Useq can be abbreviated to UQ.

Verify Command [V]

Displays the current status of Dbedit conditions.

VERIFY [ALL | @]
[BASE]
[DSET]
[SET]
[CIUPDATE]

(Default: BASE, DSET, CIUPDATE, changed SET values)

Only one item can be verified at a time. The format of the Verify output is organized into columns. Verify with no parameters prints the database, the current dataset, and whether critical-item update is enabled or disabled. It also prints those Set options which are not currently at their default setting.

Examples

`#verify dset`	{current dataset}
`#verify all`	{print the status of everything}

Welcome to Suprlink

Welcome to Suprlink

Welcome to Suprlink for MPE Version 6.1. Suprlink is a program that works with Suprtool to add "multidataset" capability to Suprtool. Rather than take the regular path to multiple datasets -- random retrieval via IMAGE keys -- with its well-known performance problems, we have chosen to follow a different path: fast serial extracts plus a very efficient merge.

Summary of the Suprlink commands:

Before	Input	Redo	Xeq
Do	Join	Reset	=expression
Exit	Link	Set	:MPE Command
Form	LISTREDO	Use	
Help	Output	Verify	

The minimum abbreviation of each command is shown in capital letters.

Installing Suprlink

Suprlink is installed as part of the Suprtool installation process. See the "Installing Suprtool" chapter of the *Suprtool User Manual* for more details of how to install both Suprtool and Suprlink.

Built-In File Names

Suprlink requires an external file for the Help command. Suprlink dynamically changes this file name, but you can use a :File command to override Suprlink's choice.

Default File Names

If you are running Suprlink on MPE V/E or MPE/iX, Suprlink finds the name of the Suprlink program file name (e.g., Suprlink.Pub.Robelle). Suprlink uses this name to determine the name of the other built-in file names. If Suprlink cannot call the procinfo intrinsic (e.g., this intrinsic doesn't exist on MPE V/R), it assumes you are running Suprlink.Pub.Robelle.

Account Name

The account name for all built-in files is the same as the one where Suprlink is running (e.g., if you :run Suprlink.Pub.Dev, the help file for Suprlink is Suprlink.Help.Dev).

Group Name

Suprlink examines the group name where Suprlink is running to determine the group name for the built-in file names. If the group name is PUB, Suprlink assumes help files are in the HELP group. The same assumption is made for any group name where the first three letters are not PUB. When Suprlink is run from a group that starts with Pub (e.g., Pub), Suprlink assumes the help files have the same suffix (e.g., Helpnew).

Examples

Here are a few examples of the names that Suprlink would use for the Suprlink help file:

Suprlink Program File Name	Suprlink Help File Name
Suprlink.Pub.Robelle	Suprlink.Help.Robelle
Suprlink.Pub.Account	Suprlink.Help.Account
Suprlink.PubRob.Account	Suprlink.HelpRob.Account
Suprlink.Robelle.Dist	Suprlink.Help.Dist

Changing Built-In File Names

You can use a :File command to tell Suprlink where the help file is located. Your :File command must use the file name that Suprlink dynamically assigned to the help file. For example, if Suprlink is called Suprlink.Robelle.Dist, you would use this :File command:

```
:file suprlink.help.dist=linkhelp.robelle.dist
```

Accessing Suprlink

How To Run Suprlink

To access Suprlink, type the following command:

```
:run suprlink.pub.robelle
SUPRLINK/Copyright Robelle Solutions Technology Inc. 1988-2001
(Version 6.1)
+
```

After a short pause, Suprlink will take over your terminal and print out some identifying information. You will notice that your command prompt has changed to "+", telling you that you have made it into Suprlink. Suprlink expects you to type command lines, ending each one with Return.

How to Xeq a Suprlink Task

Normally, you enter a series of commands. These commands specify the Input file, the Output file, and the Link file name(s). Finally, you enter an Xeq or an Exit command. This begins the actual Suprlink linkage task.

If you entered the Exit command, Suprlink will finish the current task, then return you to the Operating system.

```
+EXIT
End of program
:BYE
```

If you entered the Xeq command, Suprlink will finish the current task, then prompt you for another task. This continues until you enter the Exit command. If you wish to terminate Suprlink immediately (perhaps you are confused), enter Exit Abort. This will terminate the Suprlink program immediately, without attempting any task.

Son Process

If you RUN Suprlink within Qedit or Select, you can retain the Suprlink process and re-activate quickly later.

```
:run qedit.pub.robelle
/:r suprlink.pub.robelle
+...
+exit
SUPRLINK.PUB.ROBELLE is still alive. Okay to HOLD ? Y
/...
/:activate
SUPRLINK.PUB.ROBELLE
+...
+exit
Program Held.  Use :ACTIVATE/:RUN;HOLD to re-run.
/...
```

Suprtool Link Command

You can access Suprlink from within the Suprtool program, using Suprtool's Link command. Suprtool will then run Suprlink for you. All of Suprlink's commands are available to you in Suprtool. When passing Suprlink commands through Suprtool, you must preface each Suprlink command with *Link*. For example,

```
:run suprtool.pub.robelle
>base customer
>get m-customer
>sort custnum
>output invoices,link
>xeq
>link input invoices by custnum
>link link  customer                    {send "link customer" to Suprlink}
>link output invcust
>link exit                              {exit from Suprlink (not Suprtool)}
```

Refer to the Link command in the "Suprtool Commands" chapter of the Suprtool user manual for more information.

You can also use the Suprtool2 interface and the Link command to "call" Suprlink. See the Suprcall User Manual for a complete example of how to use the interface to pass commands to Suprlink.

Preventing MPE Commands

If you want to prevent Suprlink from executing any MPE commands (e.g., :Purge), you Run Suprlink with Parm=8192. This feature is automatically invoked by Suprtool's Link command, when Set Limit MPE Off has been specified inside Suprtool.

```
:run suprlink.pub.robelle;parm=8192
```

Exit with Verify

Some users find that they Exit from Suprlink inadvertently. For Suprlink to get user approval on Exit, Run Suprlink with Parm=64.

```
:run suprlink.pub.robelle;parm=64
>e
Okay to exit [no]:
>
```

Preventing Suprlink from Suspending

If you run Suprlink from within HPDesk (and some other programs), Suprlink will suspend on Exit but HPDesk will not notice. The next time you run Suprlink you will get a new copy of Suprlink. Eventually you will have many suspended copies of Suprlink hanging from HPDesk, consuming system resources. Running Suprlink with Parm=32 forces Suprlink to terminate on Exit rather than suspend.

```
:run suprlink.pub.robelle;parm=32
```

Job Control Word

Suprlink sets the system job control word JCW to a fatal state when Suprlink fails in a batch job. Suprlink sets only the high-order bit of the JCW job control word. That is, it adds 32,768 to the existing JCW value. HP subsystems use the other bits of the JCW job control word, so Suprlink does nothing to them.

Using Suprlink in Batch

Suprlink operates in "session" mode or "batch" mode. In batch usage, any "error" message will cause Suprlink to quit, setting the Job Control Word to flush the remainder of the JOB. Warning messages do not cause an abort.

In batch mode, Suprlink does not prompt for missing information as it does in session mode. Instead, it attempts to choose the alternative that has the least chance of destroying valid data. For example, if the Output file is a duplicate file name in batch mode, Suprlink saves the new Output file with a "made up" name (OUTPUTnn, where nn is from 00 to 20), prints a warning message, and aborts.

Summary of Parm= Values

Parm	Purpose
32	don't suspend, terminate completely
64	check with user before Exiting
8192	don't allow MPE commands

Values may be combined by adding them together. For example, Parm = 96 means "check with me before exiting, then when I do actually exit, terminate Suprlink completely instead of suspending".

SuprlinkOutCount JCW

When Suprlink closes the output file, it sets a JCW named SuprlinkOutCount with the number of output records. This is the same number reported in the total line (i.e., OUT=). You can use this JCW to control job stream execution by checking if the

SprlinkOutCount JCW is non-zero. If there are more than 65,535 output records, Suprlink sets SuprlinkOutCount to 65,535.

You can use the :Showjcw command to see the value of a JCW. For values greater than 16K, Showjcw displays either the word WARN, FATAL or SYSTEM followed by some digits. These words correspond to the following values:

```
#WARN# 16384
#FATAL# 32768
#SYSTEM# 49152
```

Add the value of the word to the number that appears with it for the true value of SuprlinkOutCount. For example,

```
:showjcw Suprlinkoutcount
SUPRLINKOUTCOUNT = WARN8616
16384 + 8616 = 25000
```

The MPE/iX :Showvar command can also be used to see the value of a JCW. Showvar displays the full, correct number (e.g., 25,000) up to the maximum of 65,535.

Suprlink also sets two other JCWs: SuprlinkOutCount1 and SuprlinkOutCount2. These communicate the full OUT= value to the Suprlink2 interface.

SuprlinkFullCount Variable

On MPE/iX, when Suprlink closes the output file, it also sets a variable namedSuprlinkFullCount with the number of output records. This is the same number reported in the total line (i.e., OUT=). The advantage of the variable is that if more than 65,535 records are written to the output file, the value of the SuprlinkFullCount variable is not truncated.

Introduction to Suprlink

How Report Programs Work

The best way to understand Suprlink is to examine the process of writing a report. Your report program will be written in COBOL, RPG, PowerHouse, or some other language. Imagine that instead of hunting all over the database with DBFIND and DBGET to collect your data, you just read a sorted disc file with a big record containing all the data on a given entity. For example, a sales report might read a disc file whose records consist of sales transactions plus customer information. This file has been sorted by customer number and date. If there are several sales for the same customer, the customer information is just repeated in each record. The report program reads the records, checks for level breaks, and formats and prints the records. Suprlink fits into this model of report programs.

Working from the database to the final flat file, how do we use Suprtool and Suprlink to produce the desired result? Obviously, Suprtool can extract the desired fields from the desired records of the sales detail dataset and put them in a disc file. And Suprtool can extract the desired fields from the customer master dataset and write them to a second disc file. What does Suprlink do?

If Suprtool sorts both files by customer, Suprlink can "link" them together, producing a third file whose composite record consists of the related fields from both files. This file is just what we need to feed into the report program.

The tests that we have performed indicate that this brute force method, which uses serial extracts, sorts, and a merge, is often significantly faster than the "official" IMAGE method of chasing down chains and hash synonyms.

Input Files

The primary Input file, which if doing a link operation is the file that should have the many or "detail" records. If any of the Link files contain duplicate records, Suprlink will select one of them to link to the primary record(s). The Suprlink Output file will have no more records than the Input file.

The Input file and Link files are created with the Output xxx,Link option in Suprtool. These files must be sorted by the same key field in ascending order.

Link Files

You can have up to seven Link files that are combined with the Input file. Suprlink merges the Input file and Link files by comparing the key fields of both files (you

can optionally specify a secondary-key). The default is for Suprlink to exclude any Input records that do not have a matching record in all Link files. Specifying the Optional keyword on the Link command will force Suprlink to fill the Output record with default values (spaces and zeros) when it doesn't find a match in a specific Link file. If you want to link the sales transaction to both the customer master and the salesman master, it's probably faster to use traditional methods (e.g., DBFIND and DBGET).

Output Files

The Output file will be a self-describing file, containing data extracted from the Input file and the Link files. Suprlink combines the Input and Link records together in a fixed way, dropping the duplicated key fields and appending the remaining fields of each file in the order specified. You control which fields occur by using the Extract command in Suprtool, but you have no control over their order. Use the Form command to print out the final record format so that you can prepare COBOL COPYLIB or PowerHouse QSCHEMA definitions.

Sort Keys

The Input file and Link files must be sorted by the same key field. Their names do not have to be identical, but they must be the same type and have the same length. Suprlink does not support real- or long-type keys.

Selection Logic

Selection logic can be tricky, since it is distributed over independent Suprtool extract tasks, the Suprlink merge phase, and the final report program.

Suprtool Selection. You can use the If command to select which records you want from each dataset. You do have the option to select key values from one dataset, then load them into a Table and use $lookup to select related entries in another dataset. It makes sense to use If on every dataset, since you have another selection possibility when the files are linked. For example, you might select all customers in California and all invoices with an amount greater than $2000.

Suprlink Selection. The Input file limits the scope of the Output file. You cannot have more Output records than you do Input records, but you can have fewer. When you do a Link to another file, you have an implied selection criterion. That is, if Suprlink cannot find a record in the Link file with matching key value(s), the Input record is dropped from the Output file. If you have seven Link files, the Input record must match all seven or be dropped. This is the default selection logic. You can override this for any specific Link file by specifying the OPTIONAL keyword on the Link command. Only do this is you don't care whether that data exists or not, since Suprlink will supply default values for those Link fields.

A Link Example

You want to produce a report of all invoices over $2,000.00 for customers in California. The customer information is in the m-customer dataset, and the invoice information is in the d-invoice dataset. Here are the steps to produce this report:

1. Select and sort the California customers into the file *customer*.

2. Select and sort invoices over $2,000 into the file *invoice*.
3. Because there will often be more than one invoice per customer, specify the invoice file as input to Suprlink.
4. Link in the customer file.
5. Produce your report from the combined records in the output file.

```
>base sales                          {sales database}
>get m-customer                      {select all customers...}
>if state = "CA"                     {...in California}
>sort custnum                        {sort and link key}
>output customer,link                {Link output option}
>xeq
```

We now have a self-describing file with all the customers from California sorted by the customer number. Next we select all invoices over $2,000.00 and sort them into customer number sequence:

```
>get d-invoice                       {select all invoices...}
>if amount>200000                    {...over $2,000.00}
>sort custnum                        {sort and link key again}
>output invoices,link                {remember the link option}
>exit
```

If we specify the cust file as input, the Output file will only contain one invoice per customer. Because we want to produce a report of all the selected invoices, we specify it as the input file:

```
+input invoices                      {driving input file, custnum is the key}
+link customer                       {combined with customers}
+output invcust                      {produces the file we want}
+exit
```

Each record of the invcust file will have both the invoice information and the customer information for each invoice of the Input file (i.e., one record per invoice). What happens if there is no customer record for a specific invoice? In this case, the invoice record does not appear in the Output file. To force Suprlink to include these records, use the optional keyword on the Link command:

```
+input invoices                      {sorted by custnum}
+link customer optional              {don't exclude invoices if...}
+output invcust                      {the customer information...}
+exit                                {...is missing}
```

A Join Example

Suprlink can join files together that have multiple key records in each file, what has been come to be know as a many-to-many link. Suprlink has traditionally been able to link an Input file with many records with the same key to a Link file that has a single record with the same key value.

The Join command will link two files with many key records in both the input file and the "Linking" file. The syntax of the Join command is exactly the same as the Link command so a sample task would look as follows:

```
+input ordhist
+join  orders
+output custord
+xeq
```

The above task will link multiple records of the file ordhist, to the multiple records of the file in orders. This assumes that the files are sorted by a common key. In SQL terms this is known as an Inner Join. An Outer Join, one where the keys do not necessarily have a match can be achieved by adding the optional keyword to the Join command:

```
+input ordhist
+join  orders optional
+output joined
+xeq
```

In SQL parlance, once again you can achieve both a Left Outer Join and Right Outer Join by reversing the order of the files, between the input and the join commands.

To give you an example of how the Join operation would work consider the following data. First we have an inventory file with multiple records for the same product-no. This data is stored in the file dinv:

```
50512001  {Rest of data}
50512001  {Rest of data}
50512003  {Rest of data}
```

The next file will have sales records, once again with multiple key values, this data is stored in the file dsales:

```
50512001  {Rest of data}
50512001  {Rest of data}
```

If you did the following task assuming both files are sorted by the product-no:

```
+in dinv
+join dsales
+out invsales
+xeq
```

The resulting file would have four records, with the multiple matching dinv and dsales records. The record layout would have the dinv information first followed by the dsales information. If you add the optional keyword on the join command the resulting file would have 5 records. The matching 4 records from dinv and dsales as well as the dinv record that did not match with the numeric fields set to zero and the byte fields set to spaces.

Only one Join operation is allowed per task.

By default, Suprlink will join files base on the primary sorted key in the self-describing file. You can specify a secondary key for the files to be joined on in a similar manner to how the Link command did:

```
+in orders
+join dsales by order-no product-no
+out ordsales
+xeq
```

Performance Considerations

Select only the records you need, unless the time to load a table of desired key values, plus the time needed to do $lookup for each record, is longer than the time to extract and sort the entire dataset. Use the Sorted and Hold options of the Table command when loading a table. Because of the time needed to search a large table, it is often faster to extract all of the records and let Suprlink skip over the ones it doesn't need.

This method does a lot of sorts. Sorting time can vary depending on system load and available memory, but it increases dramatically for large records and large datasets. You should try to use Suprtool's Extract command to reduce the record size, and consider using Suprtool's If command to reduce the number of records.

Suprlink needs enough disc space to invert a significant subset of your database, then link it into an Output file. Although all of the Suprlink files can be job temporary, you still need enough disc space for the original database, the final Output file, the primary Input file, and each of the Link files. One of the tradeoffs with this method is more disc space for faster elapsed time.

Another Example

From the sales records, retrieve all of the sales for October, 2000 and append the customer name, salesman code, and year-to-date sales total to the sales record (these fields are located in the customer records).

```
:run suprtool.pub.robelle
>base mybase
>get sales-detail                          {or use Chain command}
>extract customernum,saledate,saleamt,...
>item saledate,date,phdate
>if saledate >= $date(95/10/01) and &
>   saledate <= $date(95/10/31)
>sort customernum
>sort saledate
>output sales,link,temp                    {creates temporary SD file}
>xeq

>get customer-master                       {get entire dataset}
>extract customernum,name
>extract salesman,ytdsales
>sort customernum
>output custs,link,temp
>exit

:run suprlink.pub.robelle
+input sales                               {link sales...}
+link custs                                {...to custs...}
+output repts temp                         {...producing REPTS!}
+xeq                                       {...run the task}
+form repts                                {fields in repts}
+exit

:purge sales,temp                          {these two files...}
:purge custs,temp                          {...no longer needed}

:file infile = repts,oldtemp
:file outfile;dev=lp
:run myprog

:purge repts,temp
```

Illegal Digits

Whenever Suprlink is processing files with packed- or zoned-decimal keys, errors can occur because of invalid digits in the keys. Suprlink reports the input and link record numbers with illegal digits and processing stops. You can use Suprtool to examine input and link records, by using record selection with Suprtool's input command. A packed-decimal number consists of nibbles (there are two nibbles in each byte). The last nibble is the sign of the number. The remaining nibbles must each contain a number in the range 0-9. A zoned-decimal number must have a valid digit in each byte and end in "0"-"9", "A"-"R", "{", or "}".

Selecting Non-Matches

Consider a common problem easily solved with Quiz from Cognos: finding all records in a file which have no corresponding records in a related file. For example, to find all records in an invoice lines file with no corresponding invoice master record, the following Quiz code could be written.

```
>access lines link to header optional
>select if not record header exists
>report invoiceno of lines
>go
```

This small amount of code, however, can take a long time to execute, depending on the size of the Lines and Header files. A Quiz program will usually take longer as new links are added, causing the size of the record complex to grow.

Suprlink can provide the same information, possibly in a fraction of the time. The technique as applied to the same problem requires four steps:

1. Sort the Lines file by Invoiceno.
2. Add a new constant field, Linkflag, to the Header file and fill it with "Y". Sort by Invoiceno.
3. Link the two files with Suprlink using the Optional parameter.
4. Select the record complexes where linkflag does not contain a "Y".

```
:run suprtool.pub.robelle
>base invdb
>get lines
>sort invoiceno
>output file1,temp,link
>xeq

>get header
>define linkflag,1,1
>extract invoiceno,linkflag="Y"
>sort invoiceno
>output file2,temp,link
>xeq

>link input file1
>link link file2 optional
>link output file3,temp
>link xeq

>input file3
>if linkflag <> "Y"
>extract invoiceno
>list standard
>exit
```

Any invoice line with a corresponding record in the invoice Header file will have a "Y" in the linkflag field. Records failing the match will contain the default space.

This technique can easily be adapted for use as a command file, to minimize the amount of code added to a job stream. See the "MPE/iX Programming" section of the *Suprtool User Manual* for ideas about command files.

Linking MPE and KSAM Files

Suprlink's input or link files can come from datasets, MPE files or KSAM files. Before linking either an MPE or KSAM file, you must convert the file into a self-describing file. Because MPE and KSAM files have no implied structure, you must use Suprtool's Define and Extract commands to define the structure of the file.

Suprlink's only requirement is that the file contain a sort field and some optional data fields. The sort field does not have to be the first field in the record. The following example demonstrates how to convert an MPE file into a self-describing file:

```
:run suprtool.pub.robelle
>input mpecust                          {regular MPE file}
>def first-fields,1,10                  {fields before the sort field}
>def sort-field,11,6                    {sort field for this file}
>def last-fields,17,20                  {fields after the sort field}
>extract first-fields                   {remember to extract all of the}
>extract sort-field                     {  fields in the correct order}
>extract last-fields
>sort sort-field                        {must also remember the sort}
>output sdfile,link                     {use the Link option}
>xeq
```

This example would sort the entire MPE file. To select a subset, you would define more fields and use the If command. Treat KSAM files exactly the same way. Suprtool does not read KSAM files in sorted order, so you must remember to specify the sort explicitly when converting the KSAM file to a self-describing file.

Suprlink cannot create nor write to KSAM files. You can use Suprtool to load a KSAM file with the output from Suprlink.

CI Variable Substitution

Suprlink is able to substitute any CI variables from any command line source, whether thru interactive, use file or batch input.

In order to use this feature, first issue the following Set command:

```
set varsub on
```

Variable Substitution is not on by default for backward compatibility.

Batch

Since the Streams facility (under default setups) will replace any "!" found in the first column of a job stream. Anytime you want to specify an entire line thru Variable Substitution you will need to leave a space before the variable is specified.

```
!setvar input "i dfile by message-no"
!run suprlink.pub.robelle
+set varsub on
+ !input
```

Notes

For MPE commands some variables will be resolved twice when passed off to MPE, which will give different values for a variable.

Setting variables at the CI level:

```
MPEXL:setvar x 10
MPEXL:setvar y "!!x"
MPEXL:showvar [xy]
X = 10
Y = !x
```

Setting variables within Suprlink

```
+set varsub on
+setvar a 10
+setvar b "!!a"
+showvar [ab]
A = 10
B = 10
```

Because Suprlink does one level of variable substitution prior to the command being passed off to MPE, setting variables in Suprlink that involves other variables will give different results from MPE. We recommend that you set the variables prior to running Suprlink, or that you temporarily turn off variable substitution with the Set Varsub Off command.

```
+set varsub off
+setvar a 10
+setvar b "!!a"
+showvar [ab]
A = 10
B = !a
+set varsub on
```

Suprlink with Quiz/QTP

Quiz and QTP are part of PowerHouse, a popular fourth generation language sold by Cognos. You can use Suprtool and Suprlink to improve the performance of PowerHouse applications. For a complete discussion of how to use Suprtool and Quiz together, refer to the "Suprtool with Quiz/QTP" section of the *Suprtool User Manual*.

Suprlink can write to PowerHouse subfiles that have been created with Quiz or QTP. Subfiles are "self documenting" files that contain a complete description of the file's record structure. This information is stored in *user labels* in the file, and is known as a "mini-dictionary." When you access the subfile in Quiz, its description is read from the mini-dictionary. You must ensure that the PowerHouse subfile description *exactly* matches the record layout of Suprlink's output file. Remember that Suprlink will drop the common "key" fields from the link files.

Step 1: Create the Subfile with QTP

Before running Suprlink, you create an empty subfile with QTP:

```
:purge invcust
:qtp
>access d-invoice link custnum to &
>                     custnum of m-customer
>subfile invcust keep size numrecs include &
>        custnum, invdate, amount, invnum, &
>        name, address
>set input limit 0
>go
```

The subfile must contain all of the fields that Suprlink will produce in the output file, with the same attributes (data-type and length) and in the same order. Use the Include option of QTP's Subfile command to define each of the fields in the correct order.

The *numrecs* parameter must be replaced with the number of records that will be created by the Suprlink run. The default *numrecs* is 1023 when the input limit is set to 0.

Step 2: Output Erase in Suprlink

Once you have created the PowerHouse subfile, use the Erase option of Suprlink's Output command to load the file. This will overwrite any data in the subfile, but it will not touch the PowerHouse mini-dictionary in the user labels:

```
+input invoices                          {created by Suprtool}
+link customer                           {sorted by custnum}
+output invcust erase                    {created by QTP}
+exit
```

Step 3: Report with Quiz

The INVCUST file contains the sorted records for the Quiz report. Quiz knows the structure of this file because of the initial QTP commands that we used to create the file. Now use Quiz to generate the report:

```
quiz
>access *invcust
>report ...
>go
```

Notes on Subfiles

One of the advantages that Suprlink has over the link function in PowerHouse is that Suprlink does not require the "key" field in the link files to be a database key. Because Suprlink uses a serial-merge approach, its files only need to have a common field with the same data-type and length. If you do use Suprlink to link files that do not share a common database key, you need some extra steps to create the PowerHouse subfile.

Since Suprlink cannot currently write to NM Ksam files you cannot directly write to PowerHouse indexed subfiles. You can use Suprtool to load file to the Indexed KSAM file.

Defining Fields in QTP

In our example above, "custnum" can be used to link the d-invoice and m-customer datasets in QTP because custnum is an IMAGE key in the m-customer dataset. If custnum was not an IMAGE key, you could try declaring the record structure for the subfile with the QTP Define command:

```
:purge invcust
:qtp
>access d-invoice
>define name character size 20 = " "
>define address character size 20 = " "
>subfile invcust keep size numrecs include &
>        custnum, invdate, amount, invnum, &
>        name, address
>set input limit 0
>go
```

You must be careful to ensure that the data definitions of the Defined fields are correct. Note that you cannot assign default display specifications (such as Heading or Picture specifications) for Defined fields in QTP.

Linking Subfiles by Record Number

Another approach, which guarantees that the subfile will contain the correct data definitions and default display characteristics, is to create temporary subfiles with QTP for each dataset, then link them together by record number:

```
:purge invcust
:qtp
>access d-invoice
>subfile invtemp size 1 include &
>        custnum, invdate, amount, invnum
>set input limit 0
>go

>access m-customer
>subfile custtemp size 1 include &
>        name, address
>set input limit 0
>go

>access *invtemp link to record 0 of *custtemp
>subfile invcust keep size numrecs include &
>        custnum, invdate, amount, invnum, &
>        name, address
>set input limit 0
>go
```

Suprlink Commands

General Notes

When you run Suprlink, it prompts for commands on $stdlist with a "+" character and reads command lines from $stdinx. Suprlink commands contain a command name followed by one or more parameters, and are patterned after the same commands in Suprtool.

In this chapter, we describe the Suprlink commands in alphabetic order. Following each command name in brackets is the minimal abbreviation for the command. For example: [I] for Input and [L] for Link.

Abbreviating

You may shorten the command name to the first letter of the command name.

```
+v                                              {verify}
+x                                              {xeq}
```

Uppercase or Lowercase

You may enter the letters in either uppercase or lowercase, because Suprlink upshifts everything in the command line except literal strings within quotes ("abc"). These two commands are identical:

```
+EXIT
+exit
```

Continuation

The maximum *physical* command line is 256 characters. You may enter commands on multiple input lines by putting an "&" continuation character at the end of the line. The maximum total command length is 256 characters. The most common reason for continuing commands is to specify a lengthy Link command with secondary keys.

```
+input students
+link majors by ssn cmaj from &
                ssn curmajor
+output outfile
+exit
```

Comments on Command Lines

Comments may appear at the end of any command line, when they are surrounded by braces. Many of the examples in this manual show comments at the end of each command line. You can enter a comment as the only item in a Suprlink command

line. When continuing command lines, the comment can appear before or after the continuation character.

```
+                       {link customer records to invoices. }
+input invoices         {sorted by custnum}
+link customer          {combined with customers}
+output invcust         {produces the file we want}
+exit
```

STREAMX

STREAMX is a product from VESOFT that permits you to build flexible job streams. STREAMX contains a complete programming language with loops, prompts, and parameter substitution. A problem arises when trying to enter comments into a Suprtool batch job that will be submitted with STREAMX. Suprtool uses the {...} pair to delimit comments. STREAMX uses these same characters for expressions.

You cannot change Suprtool's comment character, but you can change the {...} characters in STREAMX. The following example changes the STREAMX expression characters from {...} to ~...~:

```
!job example,user.acct
::setbraces ~~
!run suprlink.pub.robelle
+input invoices                          {driving input file, custnum is the key}
+link customer                           {combined with customers}
+output invcust                          {produces the file we want}
+exit
!tell ~hpjobname~,~hpuser~.~hpaccount~;Example done.
!eoj
```

MPE Commands

Suprlink also accepts MPE commands, with or without a colon.

```
+:comment
+comment
```

For commands that are the same in both Suprlink and MPE, Suprlink only executes the MPE command if you type the colon. For example:

```
+help                                    {you get Suprlink help}
+:help                                   {you get MPE help}
```

MPE/iX Commands

Suprlink/iX will execute any MPE command (e.g., Run), UDCs, and command files. **Caution:** programs that suspend, instead of terminating, are not killed by the HPCICOMMAND intrinsic.

File Names

Suprlink's Input, Link and Output commands accept a file name in either MPE or POSIX format. File names starting with "/", "./", or "../" are treated as POSIX file names. All other file names are assumed to be MPE file names. MPE file names are upshifted and POSIX file names are not. POSIX file names are limited to a maximum of 240 characters. Here are some examples of POSIX file names:

```
:hello david,mgr.dev,david
:CHDIR SUBDIR
:run suprlink.pub.robelle
+input ./file
+verify input
/DEV/DAVID/SUBDIR/file
```

Calculator

Any command line beginning with an equal sign (=) is treated as a calculator expression. This feature can be used to compute blocking factors and do other calculations without the need of an electronic calculator.

You can obtain a short description of the calculator by entering the following:

```
=?                                    {? gives help}
                                      {prints a summary of = functions}
```

For a detailed description of the calculator and its options, see the Suprtool manual.

Control-Y

You can interrupt a Suprlink task with the Control-Y key (hold down Control while striking Y). Suprlink responds by telling you how far it has gotten (IN=, OUT=, etc.), and asking if you wish to stop. Hit the Return key to continue or type YES to stop the task.

Before Command [B]

Repeat any combination of the previous 1000 command lines, with or without editing.

BEFORE [*start* [/ *stop*]]
[*string*]
[ALL | @]

(Default: redo previous line)

(BQ=redo without change)

The Before command allows you to modify the commands before it executes them. If you don't need to change them, use BQ or Do.

The Before command uses Qedit-style Control characters for modifying the commands. The default mode is to replace characters. To delete use Control-D, and to insert use Control-B. If you prefer HP-style modify (D, R, I, and U), use the Redo command instead of Before.

Examples

```
+listf @.soruce                          {"source" is not spelled right}
NON-EXISTENT GROUP.  (CIERR 908)
+Before                                  {redo most recent command}
listf @.soruce                           {last command is printed}
         our                             {you enter changes to it}
listf @.source                           {the edited command is shown}
                                         {you press Return}

+listredo -10/
+before 5                                {redo 5th command in stack}
+bef 8/10                                {redo 8th through 10th}
+b listf                                 {redo last listf command}
+b listf temp                            {redo "listf temp" command}
+b @temp                                 {redo last containing "temp"}
+before -2                               {redo command before previous}
+before -5/-2                            {redo by relative lines}
```

Modify Operators

If you wish to change any characters within the line, the modify operators are the regular Control Codes used in Qedit:

- Any printing characters replace the ones above.
- Control-D plus spaces deletes columns above.
- Control-B puts you into "insert before" mode.
- Control-A starts appending characters at the end of line.
- Control-A, Control-D, plus spaces, deletes from the end.
- Control-T ends Insert Mode, allowing movement to a new column.
- Control-G recovers the original line.
- Control-O specifies "overwrite" mode (needed for spaces).

Persistent Redo

Redo commands can be saved in a permanent file and can therefore be used from another session. You can use the **Set redo** command to specify a filename to save your redo commands. Please see the Set Redo command for details.

Do Command [DO]

The Do command will repeat (without changes) any of the previous 1000 commands.

DO [*start* [/ *stop*]]
[*string*]
[ALL | @]

(Default: repeat the previous command)

Commands are numbered sequentially from 1 as entered and the last 1000 of them are retained. Use the :Listredo command to display the previous commands. You can repeat a single command (`do 5`), a range of commands (`do 5/10`) or the most recent command whose name matches a string (`do list`). If you want to modify the commands before executing them, use Redo or Before.

Examples

```
+listredo
+do                                  {do previous command again}
+do 39                               {do command line 39 again}
+do 5/8                              {do command lines 5 to 8 again}
+do link                             {do most recent Link command}
+do show                             {do last starting with "show"}
+do showjob job                      {do last "showjob job" command}
+do @job                             {do last containing "job"}
+do -2                               {do command before previous}
+do -7/-5                            {do by relative line number}
+do 5/                               {do command lines 5 to last}
```

Notes

The Do command cannot be abbreviated.

Persistent Redo

Redo commands can be saved in a permanent file and can therefore be used from another session. You can use the **Set redo** command to specify a filename to save your redo commands. Please see the Set Redo command for details.

Exit Command [E]

Exit Suprlink in one of three ways. By default, perform the current linkage task, if any, then leave Suprlink. Users are often frustrated when they exit Suprlink after specifying part of a task and Suprlink starts processing the task. Use the Abort or Suspend options to exit Suprlink conveniently without executing the current task.

EXIT [ABORT | SUSPEND | XEQ]

(Default: XEQ)

Typing Exit with no parameters means Exit Xeq. Suprlink recognizes special command names which specify both the Exit command and an exit option (e.g., ES means Exit Suspend).

Exit Abort [EA]

Cancels the current operation and terminates Suprlink. The Exit command without parameters always attempts to perform the task currently specified, while Exit Abort cancels the task and terminates immediately. Should Suprlink be executed as a son process, Exit will only suspend Suprlink, while Exit Abort will actually terminate the process.

Examples

```
+:comment.  You began to specify a linkage, stopped for
+:comment.  coffee, and decided to cancel the task
+:comment.  upon your return.
+input invoices
... coffee break ...
+exit abort                                  {cancel linkage and terminate}
```

Exit Suspend [ES]

When running Suprlink as a son process (e.g., from Qedit), it would be nice to suspend Suprlink without executing the current task. Exit Suspend does this. After returning to Suprlink, all specifications for the current task are still in effect.

Examples

```
:run qedit.pub.robelle
/:run suprlink.pub.robelle
+input invoices
+link customer                               {start specifying options}
+exit suspend                                {return to Qedit without an Xeq}
SUPRLINK.PUB.ROBELLE is still alive. Okay to HOLD ? Y
/...
/:activate suprlink
SUPRLINK.PUB.ROBELLE
+output invcust                              {continue specifications}
+xeq                                         {execute the current task}
```

Exit Xeq [EX]

To perform the current linkage task, you can either use Xeq (which leaves you inside Suprlink, ready to define another task) or Exit Xeq (which leaves Suprlink when done with the task).

After the Suprlink task completes, Suprlink either terminates, or suspends and awakens a father process (i.e., :RUN from within Qedit). Exit Xeq is the default option (i.e., specifying exit starts execution of the current task).

Examples

```
:run suprlink.pub.robelle
+exit                                   {no input was specified}
End Of Program
:run suprlink.pub.robelle
+input invoices
+link customer
+output invcust
+exit                                   {link and stop}
```

Form Command [F]

Display the fields in a self-describing file.

FORM [*filename*]

If no file name is specified, the fields in the input file are displayed. The display shows the field type and field length in IMAGE notation. An I1-field is a single integer. Packed-fields show the number of nibbles (subtract one to obtain the number of digits). Byte and zoned-decimal fields show the byte length.

When showing the form of a self-describing file, Suprlink shows the byte offset of each field after the subcount, type, and sublength. The first field always appears at offset one.

There are two types of self-describing files. One type is produced with Suprtool's Query output option. You produce the other type with the Link output option. The Form command shows the internal self-describing version number, enabling you to tell the difference.

A.00.00 - Query Output Option

Compound fields have a question mark for the type, and the length is the number of bytes in the field. Sort information about the file is missing. Here is an example form listing:

```
+form custfile
File: CUSTFILE.EXAMPLE.ROBELLE   (SD Version A.00.00)
   Entry:                Offset
      CHARACTER          X5    1         {length is five bytes}
      ZONED              Z5    6         {room for five digits}
      INTEGER            I1   11         {single integer}
      DOUBLE             I2   13         {double integer}
      PACKED             P6   17         {room for five digits}
      QUAD               I4   20         {eight-byte integer}
      REPEATINT          ?6   28         {compound field}
      LOGICAL            K1   34         {single logical}
      DBLLOG             K2   36         {double logical}
Limit: 10000  EOF: 15  Entry Length: 44  Blocking: 64
```

B.00.00 - Link Output Option

These self-describing files contain information about how the file is sorted. Compound fields are handled correctly, so the Form command shows compound fields just as you would see them in IMAGE. The Item command in Suprtool identifies the date format of an item. The Link output option saves the date format and any decimals as part of the field description:

```
+form datafile
File: DATAFILE.EXAMPLE.ROBELLE   (SD Version B.00.00)
   Entry:                Offset
      CHARACTER          X5    1  <<Sort #1 >>
      REPEATINT         3I1    6        {compound field}
      DATE               J2   12  <<YYYYMMDD>>
      DOLLAR             P6   16  << .2 >>
Limit: 10000  EOF: 15  Entry Length: 16  Blocking: 64
```

Formout File

The Form command writes all output to the file Formout. This file defaults to $stdlist. You can redirect this file to a line printer or a disc drive. If you redirect the Formout file to a disc file, Suprlink assumes a temporary file by default.

```
+:file formout;dev=lp
+form custfile                          {writes to line printer}
+:file formout;dev=disc
+form invfile                           {writes to temporary file}
```

Help Command [H]

Show what commands and options are available in Suprlink.

HELP [*command* | *keyword* [,*option*]]

(Default: browse through the entire help file)

Command Help

If you specify any parameters, Help first assumes that you want help on a specific Suprlink command. If you know the structure of the help file, you can specify one of the keywords under the command name.

`+help link`	{help on the Link command}
`+help link,notes`	{notes section of the Link command}

Keyword Help

If we cannot find any help in the "Commands" section of the help file, we assume that you specified one of the outer-level keywords in the help file. To see this list of keywords, type help with no parameters. You will see a short introduction to Suprlink and then a list of keywords. You can specify any of these keywords on the Help command. You can also specify a subkeyword.

`+help before,example`	{example section of Before command}

Quick Help - HQ

HQ asks Suprlink to look under the keyword QUICK in the help file. QUICK contains the text from the Suprlink Quick Reference Guide, offering the experienced user a quick review of the syntax of any command.

`+hq input`	{quick description of Input}
`+hq commands`	{quick list of command names}

Notes

If no parameters are specified, Help allows you to browse through the help file, Suprlink.Help.Robelle. The Help command uses the Qhelp subsystem from the QLIB. For "help in help", type "?" when you see the Qhelp prompt character ("?"). The help file is organized into levels. To go back to the previous level, press Return. Press F8 to exit the Qhelp subsystem and return to Suprlink.

Input Command [I]

Specifies the primary input source and the name of the key field by which it is sorted.

INPUT *filename* [BY *key-field*]

There can be only one Input file per linkage task, but up to seven Link files. The Input file should be created by Suprtool using the Output-Link option and must be sorted by *key-field*. The key field can be any type, except for Real or Long. The primary Input file may have more than one record per key value, and each record may appear in the Output file.

It is best to have Suprtool Extract only the fields you will actually need, since if any of the Suprtool extracts result in enormous Output files, the time to do the sort may be prohibitive.

The BY-clause is only necessary when the Input file has been created using the Suprtool Output-Query option instead of the Output-Link option. Output-Link adds the sort field information to the self-describing file, so that you do not have to specify it in a BY clause.

Join Command [J]

Join the Input file to another Join file, this links files with multiple key values in both the input file and the Join file.

JOIN *filename* [BY *join-keys* [FROM *input-keys*]]
[OPTIONAL | REQUIRED]

(Default: REQUIRED)

File Name

The Join file should be created by Suprtool with the Output,Link option; it should only contain the fields that you actually need in the final report, plus any sort fields . If you do a :LISTF of the file, it should show "SD" as the CODE; this means it is self-describing. It contains a description of its own record structure in some special user labels; this allows you to refer to the same field names as in the original database and Suprlink can compute where they occur in the record. For example:

```
+input sales                    {Sales is sorted by custno}
+join custfile                  {key is custno}
+output custsale                {Join two files...}
+exit                           {...into custsale}
```

Join Keys

Suprlink allows files to be linked by up to two keys, a primary and a secondary key field.

By default, Suprlink assumes that the key field to the Join file is the same key field specified for the Input file. If the Join key field is different from the Input key field, use the BY-clause to specify the correct key field:

```
+input customer                 {key-name is custnum}
+join sales by custno           {new name for the same field}
```

You would also use the BY-clause if the Link file was created using the Suprtool Output,Query option instead of Output,Link.

Secondary Keys

Suprlink has an option that allows you to select which join record you want by matching a second key field in the master.

JOIN *filename* BY *primary-key secondary-key*

This option forces Suprlink to compare both the primary-key and the secondary-key when comparing an input record to a join record. For example,

```
+input ordhist                  {key-name is cust}
+join orders by cust prod          {Orders contains prod}
```

This example says that the file Orders is sorted by both cust and prod fields. The join will occur on those records that match both keys.

Secondary Input Key

It is possible that the second key field has a different name in the input file and the Join file. The FROM-clause lets you handle this case:

```
+input students                              {key-name is ssn}
+join orders by ord prod from orders products
```

Note that you must specify the Input file key field as part of the FROM-clause. This example is identical to the previous secondary key example, but in this case the current major field is called "products" in the ordhist file and "prod" in the orders file.

Optional Join

If there are no join records for a given key value of the input file, that input record is dropped from the output file (this is the default option, REQUIRED).

To make the join optional, specify the OPTIONAL keyword. When you use OPTIONAL, and Suprlink does not find a matching join record in the file, Suprlink fills in the linked fields with default values. The default for byte-type fields is spaces, for zoned-type the default is ASCII zeros "0", and for all other types the default is binary zeros. For example,

```
+input custfile                              {key-name is custno}
+join addrfile optional                      {don't drop customers...}
+output custaddr                             {...if there is no address}
+exit
```

Link Command [L]

Link the Input file to another Link file, maximum of seven input files.

LINK *filename* [BY *link-keys* [FROM *input-keys*]]
[OPTIONAL | REQUIRED]

(Default: REQUIRED)

File Name

The Link file should be created by Suprtool with the Output,Link option; it should only contain the fields that you actually need in the final report, plus any sort fields. If you do a :LISTF of the file, it should show "SD" as the CODE; this means it is self-describing. It contains a description of its own record structure in some special user labels; this allows you to refer to the same field names as in the original database and Suprlink can compute where they occur in the record. For example:

```
+input sales                    {Sales is sorted by custno}
+link custfile                  {key is custno}
+link addrfile
+output custsale                {link three files...}
+exit                           {...into custsale}
```

Link Keys

Suprlink allows files to be linked by up to two keys, a primary and a secondary key field.

By default, Suprlink assumes that the key field to the Link file is the same key field specified for the Input file. If the Link key field is different from the Input key field, use the BY-clause to specify the correct key field:

```
+input customer                 {key-name is custnum}
+link sales by custno           {new name for the same field}
```

You would also use the BY-clause if the Link file was created using the Suprtool Output,Query option instead of Output,Link.

Secondary Keys

Suppose that you are linking a master to a detail and the detail can have several entries for each master. Suprlink has an option that allows you to select which link record you want by matching a second key field in the master.

LINK *filename* BY *primary-key secondary-key*

This option forces Suprlink to compare both the primary-key and the secondary-key when comparing an input record to a link record. For example,

```
+input students                 {key-name is ssn}
+link majors by ssn cmaj        {Students contains cmaj}
```

This example says that the file Majors is sorted by ssn and may contain more than one record per student. To select the desired record for each student, Suprlink matches the students' cmaj against the cmaj in the link record.

Secondary Input Key

It is possible that the second key field has a different name in the input file and the Link file. The FROM-clause lets you handle this case:

```
+input students                          {key-name is ssn}
+link majors by ssn cmaj from ssn curmajor
```

Note that you must specify the Input file key field as part of the FROM-clause. This example is identical to the previous secondary key example, but in this case the current major field is called "curmajor" in the students file and "cmaj" in the majors file.

Optional Linkage

If there is more than one link record with the same key value, Suprlink will select the first one it finds. You can sort by another value such as date-time to force a certain record to be first. Please note that this is unlike Quiz, which does a hierarchical expansion to include every record accessed. If there are no link records for a given key value of the input file, that input record is dropped from the output file (this is the default option, REQUIRED).

To make the linkage optional, specify the OPTIONAL keyword. When you use OPTIONAL, and Suprlink does not find a matching link record in the file, Suprlink fills in the linked fields with default values. The default for byte-type fields is spaces, for zoned-type the default is ASCII zeros "0", and for all other types the default is binary zeros. For example,

```
+input custfile                          {key-name is custno}
+link addrfile optional                  {don't drop customers...}
+output custaddr                         {...if there is no address}
+exit
```

Listredo Command [LISTREDO]

The Listredo command will display any of the previous 1000 commands.

LISTREDO [*start* [/ *stop*]] [;ABS] [;OUT=*file*]
[*string*] [;REL]
[ALL | @] [;UNN]

(Default: display previous 20 commands)

(BJ and ,, are short for LISTREDO)

Commands are numbered sequentially from 1 as entered and the last 1000 are retained. You can display a single command, a range of commands, all 1000, or all the commands whose name matches the string. You can print the commands with ABSolute line numbers (the default), RELative line numbers (-5/-4), or UNNumbered. You can write the commands to your terminal or OUT to a temporary file. If you want to redo any of these commands, see Do, Redo, and Before.

Examples

```
+listredo 5
+listredo 5/10
+listredo help                          {print all Help commands}
+listredo -10                           {print last ten commands}
+listredo ALL                           {print entire redo stack}
+listredo purge                         {print all Purge commands}
+listredo purge xx                      {print all "purge xx" commands}
+listredo @purge                        {print all with "purge" anywhere}
+listredo @;rel                         {print ALL, relative numbers}
+listredo 1/10;out=*lp                  {dump commands to printer}
+listredo @;unn;out=save                {write commands to a file}
```

Notes

The Listredo command cannot be abbreviated, but BJ is accepted as a short form.

Persistent Redo

Redo commands can be saved in a permanent file and can therefore be used from another session. You can use the **Set redo** command to specify a filename to save your redo commands. Please see the Set Redo command for details.

Output Command [O]

Specify the name of the output file and whether it is temporary.

OUTPUT *filename* [TEMP] [ERASE] [DATA] [LINK]

By default, the name of the Output-file is Output. The output file is a permanent, self-describing file, containing data extracted from the input file and the Link files. A few application tools may not read self-describing files. In this case, use the Data option to make the output file a standard MPE file with a filecode of zero and no user labels.

There are two different types of self-describing files. The first type is created with Suprtool's Output Query option. A superior form of self-describing file is produced with Suprtool's Output Link option. Suprlink creates the output self-describing file in the same format as the input file. We recommend that you use the same type of self-describing file for all input and link files.

Output Record Format

The record structure is determined by Suprlink, but is relatively easy to anticipate. Suprlink starts with all of the fields of the input file, in order. For each Link file, it appends the fields of the Link-file to the Output record, in order. Suprlink drops the key fields from the Link records, since they always contain duplicated data.

If a field name (other than one of the two explicit keys) is duplicated in several datasets, it will end up duplicated in the final output file. An example would be a Timestamp field that occurs in every dataset. Workaround: use the Extract command from Suprtool to take out only the fields you want, or to rename duplicate fields.

You can verify the format of the Output-file using the Form command. It shows the field names, length, and structure, in order. From this display, you can generate an appropriate COPYLIB or QSCHEMA definition.

Quiz Subfiles

The Erase option is provided for Quiz users who create an empty subfile using QTP or Quiz before running Suprtool and Suprlink. See the *Suprlink with Quiz/QTP* section for details.

Since Suprlink cannot currently write to NM Ksam files you cannot directly write to PowerHouse indexed subfiles. You can use Suprtool to load file to the Indexed KSAM file.

Size of the Output File

The output file is built with the same capacity as the input file. You can reduce the size of the output file using a :File command.

```
+:file allsales; disc=10000
+output allsales
```

Redo Command [REDO]

Enables you to modify and repeat any of the previous 1000 command lines.

REDO [*start* [/ *stop*]]
[*string*]
[ALL | @]

(Default: redo the previous command)

The Redo command allows you to modify the commands before it executes them. If you don't need to change them, use the Do command. Commands are numbered sequentially from 1 as entered and the last 1000 are retained. Use the :Listredo command to display the previous commands. You can redo a single command, a range of commands, or the most recent command whose name matches a string.

The Redo command uses MPE-style editing logic (D, I, R, U and >). The default mode is to replace characters. To delete, type DDDD under the characters to be removed. To insert, type I under the insertion spot, then the new characters. To undo your changes, type U. To append to the end of the line, use >xxx. To delete from the end of the line, use >DD. To replace at the end of the line, use >Rxxx. And to erase the rest of the line, use D>. If you prefer Qedit-style editing (Control-D, etc.), use the Before command instead of the Redo command.

Examples

```
+listf @.soruce                          {"source" is not spelled right}
NON-EXISTENT GROUP.  (CIERR 908)
+redo                                    {redo most recent command}
listf @.soruce                           {last command is printed}
          our                            {you enter changes to it}
listf @.source                           {edited command is shown}
                                         {you press Return}

+listredo all
+redo 5                                  {redo 5th command in stack}
+redo                                    {redo previous command}
+redo -2                                 {redo command before previous}
+redo 8/10                               {redo 8th through 10th}
+redo -10/                               {redo -10 through last}
+redo purge                              {redo last Purge command}
+redo purge temp                         {redo last "purge temp"}
+redo @temp                              {redo last containing "temp"}
```

Notes

The Redo command cannot be abbreviated. To save more commands, use a :File command on the file LINKREDO before you run Suprlink:

```
:file linkredo;disc=5000
:run suprlink.pub.robelle
```

Persistent Redo

Redo commands can be saved in a permanent file and can therefore be used from another session. You can use the **Set redo** command to specify a filename to save your redo commands. Please see the Set Redo command for details.

Reset Command [R]

Cancel the current linkage task.

RESET

Reset closes the current Input-file and any Link files, then resets the output file name to Output. This is actually a Reset All command; you cannot reset particular commands as you can do in Suprtool. If you try to reset an individual command, Suprlink prints a warning.

Set Command [S]

Enables or disables certain operating options within Suprlink. These options are not reset by Xeq or Reset commands.

```
SET     [MAPPED         ON|OFF]
        [REDO           filename]
        [STATISTICS     ON|OFF]
        [VARSUB         ON|OFF]
        [VARSUBDEBUG ON|OFF]
```

Mapped

SET MAPPED ON | OFF

(Initially: OFF)

MAPPED forces Suprlink/iX to read the input and link files using mapped file access. Specifying this option is an error in Suprlink/V. If the input and link files are not in memory, the wall time performance is worse with Set Mapped On, but CPU time performance is better. You must Set Mapped On before specifying the input file.

Redo

SET REDO *filename*

(Initially: unnamed temporary file)

Commands entered at the Suprlink prompt are saved in something called the redo stack. You can recall commands from the redo stack by using other commands such as Before, Do and Redo. By default, the redo stack is stored in a temporary file and discarded as soon as you exit. This temporary stack is not preserved across Suprlink invocations.

The new Set Redo command assigns a permanent file as the redo stack, allowing the stack to become available for future Suprlink invocations. For example, to assign the Myredo file as a persistent redo stack, enter

```
+set redo myredo
```

If the file does not exist, Suprlink creates it. Otherwise, Suprlink uses the existing file. All subsequent commands are written to the persistent redo stack. The setting is valid for the duration of the Suprlink session. As soon as you exit Suprlink, the setting is discarded. Next time you run Suprlink, you will get the temporary stack.

If the file name is not qualified, the redo stack is created in the logon group and account. This may be desirable if you want to have separate stacks. If you want to always use the same persistent stacks, you should qualify the name.

The Verify command shows which stack is currently in use. If it shows <temporary>, it means Suprlink is using the default stack. Anything else is the name of the file used on the Set Redo command.

Concurrency

When Suprlink uses the default temporary stack, it is only accessible to that particular instance of Suprlink. You can run as many Suprlink instances as you need

and each one gets its own redo stack. With temporary stacks, you will never get into concurrency problems.

If you start using a persistent redo stack, however, you might start running into concurrency problems. A persistent redo stack can only be used by one Qedit instance at a time. If you try to use a persistent redo stack that is already in use, you will get the following message:

```
+set redo myredo
EXCLUSIVE VIOLATION: FILE BEING ACCESSED  (FSERR 90)
Unable to open file for REDO stack
```

In this situation, Suprlink continues to use the redo stack active at the time and lets you continue working as normal.

Qedit can also have permanent redo stacks. To prevent products from writing to each other's stack, it is advisable to have separate stacks for each product by giving them different file names.

For example, if you use

```
set redo myredo
```

you will have a redo stack called Myredo for your Suprlink commands. If you exit Suprlink, then run Qedit and supply the same Set Redo command, your Qedit commands will be written to the same file that was used for your Suprlink commands.

Statistics

SET STATISTICS ON | OFF

(Initially: OFF)

STATISTICS causes Suprlink to print statistics at the end of each task.

Varsub

SET VARSUB ON | OFF

(Initially: OFF)

Setting Variable Substitution causes Suprlink to resolve any CI variables in a command before processing.

VarsubDebug

SET VARSUBDEBUG ON | OFF

(Initially: OFF)

Suprtool, now has a setting called Set VarsubDebug on which will print out the line after the variable substitution has occurred. This setting only works if Set Varsub is on and Set VarsubDebug is on.

```
setvar outfile &
:"/GREEN/SUPRTEST/filename901234567890123456789012345678901234567890123
45678901"
:run suprtool.pub.robelle
SUPRTOOL/iX/Copyright Robelle Solutions Technology Inc. 1981-2007.
(Version 6.2 Internal)  TUE, OCT 30, 2007,  2:58 PM  Type H for he
>set varsub on
>set varsubdebug on
>in file1sd.suprtest
vd:in file1sd.suprtest
>output !outfile,link,temp
vd:output /GREEN/SUPRTEST/filename90123456789012345678901234567890123
vd:2345678901,link,temp
```

The output is formatted into 74 byte chunks and printed with a preceding “vd:” so the “substituted” line is clear.

Use Command [U]

Specifies a file of commands to be executed as a group.

USE[Q] *filename*

Examples

A usefile makes your task easier by allowing common commands to be specified once in an external file. For example, the following usefile contains all the commands for creating the invcust file:

```
+use usecust
input invoices                    {sorted by custnum}
link customer                     {combined with customers}
output invcust                    {produces the file we want}
exit
```

Suprlink prints the lines in the usefile, including the comment lines. This allows you to include instructions and reminders in the usefile. In the example above, there were no commands for the user to enter.

Notes

Usefiles cannot be nested in Suprlink. The usefile may be any unnumbered text file or a Qedit workfile, but no more than 256 characters per record will be processed.

By default, Suprlink displays the commands in a usefile as they are executed. Suprlink can execute commands *quietly* using the Useq command. For compatibility with Qedit, Useq can be abbreviated to UQ.

Verify Command [V]

Print the definition of the current linkage task.

VERIFY

Verify prints the current Input, Link, and Output files; in other words, it is a Verify All command.

Xeq Command [X]

Perform the current linkage task.

XEQ

Xeq checks that you have specified an input file and at least one Link file. Then it performs the linkage and creates the output file. Finally, it closes the files and resets, ready for you to specify another linkage task or Exit. If you also wish to leave Suprlink after completing the linkage task, use Exit instead of Xeq.

Example Suprlink Output

Example

The Form command displays the fields in a self-describing file. This information is stored in the user labels of an MPE file and is not accessible with other tools. Use the Form command to obtain the record layout of Suprlink output files.

The following example shows the Form command listing for an input file, a Link file, and the resulting output file. We start with an input file of invoices.

```
+form invoices
File: INVOICES.TEST.DATA   (SD Version B.00.00)
   Entry:                     Offset
      CUSTNUM               X8      1  <<Sort #1 >>
      DELIVERED             I2      9
      PRODUCTNUM            Z8     13
      PRICE                 I2     21
      PURCHASED             I2     25
      QTY                   I1     29
      TAX                   I2     31
      TOTAL                 I2     35
Limit: 100  EOF: 100  Entry Length: 38  Blocking: 107
```

Suprtool produced both the invoice and the customer file by using the Get, Extract, and Sort commands. The invoice file was produced with Suprtool's Output Link option. If you had used Suprtool's Output Query option, the Form command would not have printed any information about the key fields. The next listing is the customer file.

```
+form cust
File: CUST.TEST.DATA   (SD Version B.00.00)
   Entry:                     Offset
      CITY                  X12     1
      RATING                I2     13
      CUSTNUM               X8     17  <<Sort #1 >>
      STATUS                X2     25
      FIRSTNAME             X10    27
      LASTNAME              X16    37
      STATE                 X2     53
      ADDRESS              2X25    55
      ZIPCODE               X6    105
Limit: 50  EOF: 50  Entry Length: 110  Blocking: 37
```

The street address is a compound-field. If you had used Suprtool's Output Query option, the field would have appeared with a question mark for the data-type. In that case, you cannot use the field as a key-field in Suprlink, but the actual data in the field will be processed and linked correctly. Your final report should be able to read this data just as if it came from the database. We use Suprlink to combine the invoice and cust files into one Output-file:

```
:run suprlink.pub.robelle
+i invoices by custnum
+l cust
+o invcust
+e
```

The final Form command shows the record layout of the Output-file. You would use this file as input to your report program.

```
+form invcust
File: INVCUST.TEST.DATA   (SD Version B.00.00)
   Entry:                     Offset
      CUSTNUM               X8      1  <<Sort #1 >>
      DELIVERED             I2      9
      PRODUCTNUM            Z8     13
      PRICE                 I2     21
      PURCHASED             I2     25
      QTY                   I1     29
      TAX                   I2     31
      TOTAL                 I2     35
      CITY                  X12    39
      RATING                I2     51
      STATUS                X2     55
      FIRSTNAME             X10    57
      LASTNAME              X16    67
      STATE                 X2     83
      ADDRESS              2X25    85
      ZIPCODE               X6    135
Limit: 100  EOF: 100  Entry Length: 140  Blocking: 29
```

Limits Within Suprlink

Maximums

The various limitations of Suprlink are described here. In general you need to reduce the number or sizes of fields if you encounter any of these limits.

Input File - Maximum Record Size - 4096 Words

This is also the maximum size of an IMAGE entry. We recommend that you use Suprtool's Extract command to minimize the input record size.

Input File - Maximum Block Size - 4096 Words

By default, Suprtool restricts the maximum block size to 2,048 words. You can use the Set Blocksize command to increase this size up to 8192 words. If you increase the maximum block size, it is likely that Suprtool will produce an output file that Suprlink cannot read.

Input File - Maximum Fields - 255

Suprlink restricts the number of fields per file to be 255. If you must have more fields, use Suprtool's Define and Extract commands to extract several fields as one contiguous series of bytes.

Link File - Maximum Record Size - 4096 Words

As with the input file, you should use Suprtool's Extract command to minimize the link record size.

Link File - Maximum Block Size - 2048 Words

See the description of the maximum input block size.

Link File - Maximum Fields - 255

See the description of the maximum number of input fields.

Link File - Maximum Number - Seven

Suprlink will link one input file with up to seven Link files.

Output File - Maximum Record Size - 8192 Words

When linking many files together, it is easy to produce large output records. Once again, using the Extract command to minimize the size of the input and link records will avoid large output records.

Output File - Maximum Fields - 1023

Internal Suprlink tables that keep track of the output fields are restricted to 1023 entries.

Welcome to Speed Demon

Welcome to Speed Demon

Welcome to version 6.2 of Speed Demon. For improved performance, call Speed Demon in your programs instead of DBGET mode-2 to read serially through a large dataset.

There are a few restrictions to using Speed Demon:

1. The database must not be open in mode-3 or mode-7.
2. You must have read access to all of the fields in the dataset.
3. The field list cannot be the "same" list (i.e., no *-list).
4. The database must be local (i.e., not on a remote machine).
5. Entries cannot be updated or deleted.
6. On MPE V (or compatibility mode on MPE/iX), the calling program must not use DB-14 and DB-32 of the data stack.

MPE V

On MPE V, you install Speed Demon in the System SL (see *Installing Speed Demon* for instructions). Speed Demon/V should be about five times faster than DBGET mode-2.

MPE/iX

On MPE/iX, you run your native mode programs with an XL file: DemonXL.Pub.Robelle. Speed Demon/iX can be slightly slower than DBGET mode-2 or it can run as much as eight times faster. We are not certain why there is such a range of performance values. We would be happy to see performance results from users.

Conditions of Use

Speed Demon is included as part of Suprtool. Any licensed Suprtool users can use Speed Demon in their own application software. Speed Demon cannot be included in software being distributed to any sites that are not licensed to use Suprtool. Application software developers who would be interested in integrating Speed Demon into their products should contact Robelle Solutions Technology Inc.

Speed Demon vs. Suprtool

Suprtool is a program while Speed Demon is a set of callable routines. Application programs can *invoke* Suprtool to extract, select, and sort records from an IMAGE dataset, producing a disc file as output. Application programs can *call* Speed Demon to serially return the individual records of the same IMAGE dataset, one record at a time.

While Speed Demon lacks Suprtool's ability to extract fields or select and sort records, it has the advantage that it delivers the records directly into your program. Speed Demon/V is about 50% slower than Suprtool, but there is no need to read an output file, as with Suprtool. Speed Demon/iX runs at the same speed as Suprtool. With Speed Demon, your application program is responsible for selecting records and sorting them (e.g., using the SORT verb in COBOL).

Application programs should invoke Suprtool when they expect to select a *small percentage* of the records (i.e., when the overhead of reading the OUTPUT file will be minimized). Application programs should call Speed Demon when they expect to select a *large percentage* of the records.

Fourth-Generation Languages (4GLs)

Speed Demon will be more difficult to use from 4GLs than from a regular programming language (e.g., COBOL or Pascal). Remember: each call to Speed Demon returns a single record and you must build a loop to read and test them. Some 4GLs do a LOADPROC for every call to the Speed Demon intrinsics (e.g., Reactor from Speedware). Because LOADPROC is so slow, this eliminates any benefit from Speed Demon. We would like to hear from any users who have success using Speed Demon from a 4GL.

Caveats for Privileged Mode Users

Speed Demon/V uses split-stack mode. This should not concern any normal user, but there are two consequences for privileged mode users:

1. You cannot call Speed Demon/V from split-stack mode.
2. Do not enable a privileged mode Control-Y trap handler while calling Speed Demon/V.

SPDEPREFETCH JCW

Speed Demon can read data directly from disc into memory using Multi-Rec NOBUF reads. However, Speed Demon is often slowed down on MPE/iX while waiting for the file system to satisfy its read requests. Using prefetch on MPE/iX, Speed Demon is able to increase its throughput by instructing MPE/iX's memory manager to read the data from disc to memory ahead of time. This way, when Speed Demon needs the data it is already in memory and Speed Demon doesn't have to wait.

The SPDEPREFETCH JCW tells the memory manager how far ahead of Speed Demon it should fetch data. Setting this number too low won't give the memory manager sufficient time to get the data into memory before Speed Demon needs it. Setting the number too high may mean that on a busy system the data will be overwritten by something else before Speed Demon gets a chance to use it.

When reading an input dataset, Speed Demon/iX prefetches twice the buffer length of data from disc to memory. This default value of 2 works well, even on small machines. If Speed Demon is running stand-alone on a fast CPU with a lot of memory, you may wish to increase the prefetch amount. The maximum value for SPDEPREFETCH is 5 (i.e., five times the buffer length). If you don't want Speed Demon to prefetch, you can specify SPDEPREFETCH 0. This may be required when Speed Demon is operating with third-party software tools that intercept all file system calls (e.g., Netbase from Quest Software).

The SPDEPREFETCH JCW is checked by Speed Demon/V, but its value has no effect.

Installing Speed Demon

Compatibility and Native Mode Versions

There are two versions of Speed Demon. The Speed Demon/V version is for MPE/V systems and for compatibility-mode programs on MPE/iX. Speed Demon/V runs in compatibility mode and must be installed in the system SL. The Speed Demon/iX version is for native-mode programs on MPE/iX. Speed Demon/iX runs in native mode and is accessed via an XL file.

Depending on what operating system you are on (MPE/V or MPE/iX) and what kind of programs will be calling Speed Demon (compatibility mode, native mode or both), you may end up installing either, both, or neither version. Read all the instructions carefully.

Before using either version of Speed Demon, you must build or upgrade the Robelle account and restore all files as described in the *Installing Suprtool* chapter of the Suprtool User Manual.

System SL Installation

This procedure is required for MPE/V systems, and for compatibility mode programs on MPE/iX.

The Speed Demon/V routines are distributed in the file DemonUSL.Pub.Robelle. The job stream Demon.SuprJob.Robelle installs the Speed Demon/V intrinsics in the System SL. This is the only method for installing Speed Demon/V. You also use this job stream to update Speed Demon/V when you receive a new version, or to re-install the interface after a MIT update from HP. You will need a small tape for a new cold load tape to contain the Speed Demon/V segments.

Warning: You must have created the Robelle account and restored all files as described in the installation chapter of the Suprtool User Manual. To install Speed Demon/V into the System SL, follow these steps:

1. Ensure that no one will use Speed Demon/V until the installation is complete. No one can be running a program which uses Speed Demon/V. Stop all jobs and send an operator warning.

```
:showjob
:warn @;please stop for 20 minutes
:abortjob #snnn
```

2. Stream the installation job. If MPE prompts for passwords, supply them.

```
:stream demon.suprjob.robelle
```

3. Speed Demon/V uses the SEGMENTER to add the segments into SL.Pub.Sys. It then requests a tape ("COLDLOAD") to create a new cold load tape containing MPE plus the Speed Demon/V intrinsics. Mount a tape with a write ring and :REPLY. Save this tape and use it for any future cold loads.
4. If you're installing to an MPE/iX machine, the job does not create a cold load tape. You must create a system load tape manually.
5. If everything goes well, Speed Demon/V prints a final message on the console.
6. Please save the job listing for future reference.

Speed Demon/V is now installed and you should be able to use it in your application programs.

User XL Installation

This procedure is for native mode programs on MPE/iX, and is optional.

You normally access Speed Demon/iX by specifying XL="DemonXL.Pub.Robelle" when running your program. Therefore, there may be no need to perform any installation steps for Speed Demon/iX.

The advantage of leaving Speed Demon/iX in the Robelle account and always pointing to the XL when you run your programs, is that when you receive an upgrade of the Robelle account files, Speed Demon/iX is automatically upgraded with no effort on your part. The disadvantage is that you must remember to always put the XL= statement on all Run commands for your programs that use Speed Demon/iX. If you don't want to change your Run commands, you can copy Speed Demon/iX into your own XL that is already being searched.

Installing Speed Demon/iX in your own XL requires that both the account and the group where the XL resides have privileged mode capability. Calling programs do not have to have privileged mode capability. The XL can reside anywhere on your system, it does not have to be in the same account or the same group as your XL file. Here are the commands to install Speed Demon/iX in your own XL:

```
:hello user.acct
:linkedit
>xl xl              {we assume the XL already exists}
>copyxl from=demonxl.pub.robelle; replace
>exit
```

The Replace option is not in all versions of Linkedit. If your version does not allow Replace, and you already have Speed Demon in your XL file, you will need to manually purge the existing modules before copying the new one.

Demonxl can successfully be installed into the System XL file, but this is not recommended by HP. Demonxl can be combined into an XL file with other Robelle XL files, except Qcompxl (a part of Qedit).

Alternatively, you can specify the XL for a program to use with the Altprog and Link commands in Linkedit. To add an XL to a program that you are building with the Link command, you can specify the following from within Linkedit:

```
:linkedit
HP Link Editor/iX (HP30315A.06.14) Copyright Hewlett-Packard Co
LinkEd> link from=main; to=myprog; xl=demonxl.pub.robelle
```

To alter an existing program to use the Demonxl, you can use the Altprog command from within Linkedit:

```
LinkEd>altprog myprog; xl=demonxl.pub.robelle
```

Please see the Linkedit manual for details about the Link and Altprog commands in Linkedit.

Group or Pub SL Installation

This procedure is for MPE/V systems and for compatibility mode programs on MPE/iX. It is not the recommended or supported procedure. See the Speed Demon/V instructions in the previous section.

We only support Speed Demon/V when it is installed in the System SL. These are the reasons for this policy:

1. Speed Demon/V requires privileged mode. When Speed Demon/V is installed in the System SL, it can obtain privileged mode for itself while guaranteeing that privileged mode is not available to the calling program. If Speed Demon/V is installed in a Pub or Group SL, the calling program is installed in a group and account with privileged mode.
2. Some third-party software intercepts all IMAGE and file system calls (e.g., DBOPEN and FOPEN). Writing intercept calls is straightforward from a Pub or Group SL, but very difficult inside the System SL. Emulating all IMAGE and file system calls correctly is hard, and can introduce problems for Speed Demon/V. By installing Speed Demon/V in the System SL, we ensure that Speed Demon/V is calling the real IMAGE and file system intrinsics.

Netbase

Netbase is a product from Quest software that intercepts IMAGE and File System calls. Netbase allows any database or file to be automatically directed to a remote machine. Netbase does not support System SL installation. For this reason, the only way to make Speed Demon/V and Netbase work together is to install Speed Demon/V in a Pub SL along with the Netbase software.

Pub SL

Installing Speed Demon/V in a Pub SL requires that both the account and the Pub group have privileged mode capability. The calling program must be installed in the same account (it does not have to be in the Pub group) and it must be run with Lib=P. Here are the commands to modify an existing account called Dev.

```
:hello manager.sys
:altacct dev;cap=...,pm
:hello mgr.dev,pub
:altgroup pub;cap=...,pm
:segmenter
-buildusl $newpass,400,8
-auxusl demonusl.pub.robelle
-copy segment,suprrobelle0
-copy segment,suprrobelle1
-buildsl sl,400,8
-addsl suprrobelle0
-addsl suprrobelle1
-exit
```

The previous example shows the capability list as "cap=...,pm". You must fill in the "..." by obtaining the existing capabilities for the Dev account and the Pub group. On MPE/iX, you can use the :listacct and :listgroup commands. On other versions of MPE, you will need to :run Listdir5.Pub.Sys.

Group SL

The steps for installing Speed Demon/V in a group SL are the same as for the Pub SL, except that you use the actual group name instead of the Pub group. Your user programs must be installed in the same group as the SL and they must be run with Lib=G.

Accessing Speed Demon

Accessing Speed Demon

Speed Demon is a set of routines, or intrinsics, that you call from your program. The Speed Demon/V intrinsics are installed in the system SL. The Speed Demon/iX intrinsics are installed in an XL. You access Speed Demon by calling these intrinsics, just as you would the IMAGE intrinsics. There are three primary intrinsics:

SPDEDBINIT. Initializes the Speed Demon environment and decides what dataset to read.

SPDEDBSCAN. Replaces calls to DBGET mode-2 and returns the same condition word values (e.g., 0 for okay, 11 for end-of-file).

SPDEDBSHUT. Cleans up the Speed Demon environment. Before you can call SPDEDBINIT for another dataset, you must call SPDEDBSHUT.

Original Program Using DBGET

Suppose that your program scans the d-sales dataset to produce a report of all d-sales records with a sales-total greater than $10,000.00. Currently, your COBOL program looks like this:

```
05-report-control          section.
    move false-value to end-of-d-sales-flag.

    perform 10-read-and-report
       thru 10-read-and-report-exit
      until end-of-d-sales.

05-report-control-exit.  exit.

10-read-and-report         section.

    perform 10-10-get-d-sales.

    if not end-of-d-sales then
        if dsa-sales-total > 10000 then
            perform 20-report-d-sales
               thru 20-report-d-sales-exit.

    go to 10-read-and-report-exit.

10-10-get-d-sales.
    call "DBGET" using db-base
                       db-set-d-sales
                       db-mode2
                       db-status-area
                       db-all-list
                       db-buffer-d-sales
                       db-dummy-arg.
    if db-end-of-file then
        move true-value to end-of-d-sales-flag
    else
    if not db-stat-ok then
        perform 99-image-fatal-error.

10-read-and-report-exit.  exit.
```

Modified Program Using Speed Demon

You would modify the original code to use Speed Demon to serially read through the d-sales dataset. The original report-control section had no initializing paragraph because we assumed that the d-sales dataset was rewound to the beginning. The modified code must call SPDEDBINIT before reading the dataset. When we have finished reading the dataset, we finish off with a call to SPDEDBSHUT.

```
05-report-control          section.
    perform 05-05-init-d-sales.

    move false-value to end-of-d-sales-flag.

    perform 10-read-and-report
       thru 10-read-and-report-exit
      until end-of-d-sales.

    perform 05-20-close-d-sales.

    go to 05-report-control-exit.

05-05-init-d-sales.
    call "SPDEDBINIT" using db-base
                            db-set-d-sales
                            db-mode1
                            db-status-area
                            spde-db-control
                            db-dummy-arg.
    if not db-stat-ok then
        perform 98-spde-error.

05-20-close-d-sales.
    call "SPDEDBSHUT" using db-base
                            db-set-d-sales
                            db-mode1
                            db-status-area
                            db-dummy-arg.
    if not db-stat-ok then
        perform 98-spde-fatal-error.

05-report-control-exit.  exit.
```

The read-and-report section has one change. Calls to DBGET are replaced with calls to SPDEDBSCAN. If SPDEDBSCAN fails, we perform a different error section. Note that the parameter list for SPDEDBSCAN is not the same as for DBGET. Our original program used the @-list, so we don't need to make any changes to the buffer-d-sales declaration.

```
10-read-and-report         section.
    perform 10-10-read-d-sales
    if not end-of-d-sales then
        if dsa-sales-total > 10000 then
            perform 20-report-d-sales
               thru 20-report-d-sales-exit.

    go to 10-read-and-report-exit.

10-10-read-d-sales.
    call "SPDEDBSCAN" using db-base
                            db-status-area
                            db-buffer-d-sales
                            db-dummy-arg.
    if db-end-of-file then
        move true-value to end-of-d-sales-flag
    else
    if not db-stat-ok then
        perform 98-spde-error.

10-read-and-report-exit.  exit.
```

If the original program used partial field lists, we would call SPDEDBINIT with mode-2 and pass the field list as the last parameter. For example,

```
    move "CUST-ACCOUNT,PRODUCT-NO,PRODUCT-PRICE,PURCH-DATE;"
       to db-list-d-sales.
    call "SPDEDBINIT" using db-base
                            db-set-d-sales
                            db-mode2
                            db-status-area
                            spde-db-control
                            db-list-d-sales.
    if not db-stat-ok then
       perform 98-spde-error.
```

How to Use Speed Demon

To replace the DBGET mode-2 calls with calls to SPDEDBSCAN we did the following:

1. Called SPDEDBINIT to initialize the d-sales dataset.
2. Replaced calls to DBGET with calls to SPDEDBSCAN and modified the parameters passed to SPDEDBSCAN.
3. Performed a different error section for Speed Demon errors.
4. Called SPDEDBSHUT to close the dataset.

SPDEDBINIT and the Control Record

SPDEDBINIT requires a special control record. For COBOL, we suggest that this control record be placed in the COPYLIB and copied into programs that use Speed Demon. A common error in using Speed Demon is typing the control record incorrectly. Use the file Cobol2.Qlibsrc.Robelle instead. You can copy the control record directly into your COBOL program with the following Qedit commands (use /JOIN with EDIT/3000):

```
/add 50.1=cobol2.qlibsrc.robelle 1/20
```

The definition of the control record, with the proper initializing values is:

```
01  spde-db-control.
    05  spde-version     pic s9(4) comp value 0.
    05  spde-buffer-size pic s9(9) comp value zeros.
    05  spde-future      pic x(20) value spaces.
```

Error Handling

Programs that call IMAGE intrinsics usually have an error section for fatal IMAGE errors. These sections typically call DBEXPLAIN and abort the program. When a fatal Speed Demon error occurs, SPDEEXPLAIN should be called instead of DBEXPLAIN.

```
98-spde-fatal-error        section.
    call "SPDEEXPLAIN" using db-status-area.

    call "QUIT" using \db-status-area\.

98-spde-fatal-error-exit.  exit.
```

MPE/iX

To access Speed Demon/iX from a native mode program, you must modify the :Run command for your program to include an XL file. The following example shows how you would compile, link, and run a native mode COBOL program that calls Speed Demon/iX:

```
:cob74xl testsrc.demon
:link
:run $oldpass;xl='demonxl.pub.robelle'
```

Speed Demon Intrinsics

The Intrinsics

In this chapter we describe the Speed Demon intrinsics in alphabetic order. The parameters to each intrinsic are word arrays (just like IMAGE). You must include every parameter or pass a dummy variable as a place holder. The condition code is not returned by any Speed Demon intrinsic.

The Status Array

The status array is the same communications area that you use with the IMAGE intrinsics. If an error occurs, you should call the SPDEEXPLAIN intrinsic. The Speed Demon intrinsic does not return exactly the same status array as DBGET.

SPDEEXPLAIN Intrinsic

Prints a three-line message on $stdlist which describes the results of a Speed Demon intrinsic call based on information in the status array.

SPDEEXPLAIN *status*

Status

The twenty-byte array used to call any Speed Demon intrinsic which SPDEEXPLAIN will describe. The database and dataset parameters must not change between the call to the Speed Demon intrinsic and the call to SPDEEXPLAIN.

The Speed Demon intrinsics can return many of the standard IMAGE error numbers, but Speed Demon has many of its own. SPDEEXPLAIN will print the correct error message in either case. Because the dataset parameter is not passed to the Speed Demon intrinsic, SPDEEXPLAIN may not be able to print the name of the dataset in the second line of the message.

SPDEDBINIT Intrinsic

Initialize Speed Demon, specify what dataset to read, and what field list to use.

SPDEDBINIT *Database, Dataset, Mode, Status, Control, List*

Database

Database name as returned from DBOPEN (first two characters must be non-blank).

Dataset

Dataset name or dataset number to read. You must have full-read access to this dataset.

Mode

SPDEDBINIT can be called in any of four modes:

Mode	Function
0	Return the version number of SPDEDBINIT. All the other parameters are ignored, and Speed Demon is not initialized.
1	Initialize Speed Demon to sequentially access the dataset of the specified database.
2	Same as mode-1, but allows partial field lists.
3	Same as mode-2, but Speed Demon will read to the physical end of the dataset. Use this mode if your program expects changes to the dataset at the same time Speed Demon is reading it.

Status

If SPDEDBINIT succeeds with mode-0, the following values are returned in the status array.

Word	Usage
1	Zero.
2	Major version number.
3	Minor version number.
4	Pre-release version number.
5-10	Zeros.

If SPDEDBINIT succeeds with mode-1, mode-2, or mode-3, the following values are returned in the status array.

Word	Usage
1	Zero.
2	Entry length of the dataset.
3-4	Buffer size used to read the dataset.
4-5	Number of records that Speed Demon will read. For mode-3, this is always the capacity of the dataset. For mode-1 and mode-2, this is the number of entries for master datasets and the highwater mark for detail datasets.
7-10	Zeros.

Control

Special control record that must be declared and initialized as follows:

```
01  s7pde-db-control.
    05  spde-version     pic s9(4) comp value 0.
    05  spde-buffer-size pic s9(9) comp value zeros.
    05  spde-future      pic x(20) value spaces.
```

Normally you won't be changing this control buffer. You might want to change the spde-buffer-size. This sets the size of the file system reads and the size of the Speed Demon extra data segment. The default size is 14,336 words, but you may wish to reduce it.

List

For mode-0 or mode-1 this is a dummy parameter. For mode-2 or mode-3, this is the field list that corresponds to the buffer that will be returned by SPDEDBSCAN.

Highwater Mark

When you specify a detail dataset, Suprtool and Speed Demon read records until the highwater mark. If there is a big difference between the number of entries in the dataset and the highwater mark, Suprtool and Speed Demon could take much longer to read the dataset (this is true for all programs, not just Suprtool and Speed Demon). Because mode-1 and mode-2 calls to SPDEDBINIT return the highwater mark for detail datasets in the status area (this is the record limit), you can write your own programs to automate checking for excessive highwater marks.

Notes

You must call DBOPEN before calling SPDEDBINIT. This is just like IMAGE -- you cannot call DBGET unless you have first called DBOPEN.

When you call SPDEDBINIT, Speed Demon determines how many records to read. For master datasets, mode-1 and mode-2 of SPDEDBINIT uses DBINFO to obtain the number of entries in the dataset. For detail datasets, mode-1 and mode-2 of SPDEDBINIT uses the highwater mark for the dataset. SPDEDBSCAN stops reading the dataset when it has read that number of entries. If the dataset is changing dynamically, new records might be missed. Use mode-3 to force Speed Demon to read to the physical end of the dataset. This mode makes Speed Demon more reliable, but slower when the dataset is mostly empty.

SPDEDBSCAN Intrinsic

Read the next entry in the dataset.

SPDEDBSCAN *Database, Status, Buffer, Dummy*

Database

Database name as passed to DBOPEN.

Status

If SPDEDBSCAN succeeds, the following values are returned in the status array. The forward and backward pointers are not returned.

Word	Usage
1	Zero.
2	Entry length of the record returned in buffer.
3-4	Record number of the record read.
5-10	Zeros.

Buffer

Array where the values of the record read are placed. This buffer must correspond to the field list specified in SPDEDBINIT.

Dummy

This is a dummy parameter and it may contain anything.

SPDEDBSHUT Intrinsic

Close the Speed Demon environment. This must be done if another dataset will be read, or to restart at the beginning of the same set. SPDEDBSHUT will optionally print statistics about the read operation.

SPDEDBSHUT *Database, Dataset, Mode, Status, Dummy*

Database

Database name as passed to DBOPEN.

Dataset

Dataset name or dataset number to shut. This must be the same dataset name or dataset number passed to SPDEDBINIT.

Mode

SPDEDBSHUT can be called in any of three modes:

Mode	Function
0	Return the version number of SPDEDBSHUT. All the other parameters are ignored, and Speed Demon is not closed.
1	Close the Speed Demon environment.
2	Close the Speed Demon environment and print a summary of the read operation on $stdlist.

Status

If SPDEDBSHUT succeeds with mode-0, the following values are returned in the status array.

Word	Usage
1	Zero.
2	Major version number.
3	Minor version number.
4	Pre-release version number.
5-10	Zeros.

If SPDEDBSHUT succeeds with mode-1 or mode-2, zeros are returned in the status array.

Dummy

This is a dummy parameter. It may contain anything.

The Demon Program

The Demon-stration Program

Speed Demon includes a program for demonstration and verification. This program is called Demon.Pub.Robelle. The Demon program requests a field list to process, and accepts all valid IMAGE field lists (except for *-list "same list"). You can use this feature to compare the speed of DBGET to Speed Demon. There are several ways of accessing this program.

Reading with Speed Demon (Parm=0)

If you run the Demon program with no parm-value and no entry-point, you are prompted for a database and a password. To terminate the program, hit Return at the database prompt. Demon attempts to open the database with mode-5. If this succeeds, you are prompted for a dataset name. After prompting for the dataset name, Demon prompts for a field list. Demon uses SPDEDBINIT to verify that the dataset name and field list are valid and to initialize Speed Demon. Speed Demon accepts all valid IMAGE field lists, including partial lists, except for the same list "*". Demon then prints the entry length and buffer values returned from SPDEDBINIT.

Next you will be prompted for an output file name. You may specify any valid MPE file name, or Return for no output. The Demon program uses buffered I/O to write to the output file. Do not compare the cpu or wall time of the Demon program with Suprtool, because Suprtool will always be faster. After the Demon program has read the dataset, the SPDEDBSHUT statistics are printed on $stdlist.

Reading with DBGET (Parm=1)

If you run the Demon program with Parm=1, you will be prompted for a database, password, dataset, field list, and output file. The difference from Parm=0 is that the Demon program uses DBGET mode-2 to read the dataset instead of using Speed Demon. Use Parm=1 to compare the speed of DBGET versus that of Speed Demon.

Reading All IMAGE Blocks (Parm=3)

Running the Demon program with Parm=3 is the same as Parm=0, but SPDEDBINIT is called with mode-3. This forces Speed Demon to read to the

physical end of the dataset. Use this feature to compare the speed of SPDEDBINIT mode-2 versus mode-3.

Verifying Version Numbers

The Demon program includes a special entry point for verifying the Speed Demon version numbers. There are two Speed Demon version numbers: one for SPDEDBINIT and one for SPDEDBSHUT. The Demon program prints two sets of these version numbers: one set for the Demon program routines and one set for the System SL. The Demon program will notify you if Speed Demon has not been installed in the System SL. For example:

```
:run demon.pub.robelle,version
  Speed Demon/Copyright Robelle Solutions Technology Inc.
  (Version 6.2)

 Demon program Internal Version Numbers:
   SPDEDBINIT (Version 6.2)
   SPDEDBSHUT (Version 6.2)

 Speed Demon System SL Version Numbers:
   SPDEDBINIT (Version 6.2)
   SPDEDBSHUT (Version 6.2)
```

Demon/iX does not print the System SL version numbers. If you are on MPE/iX and have installed Speed Demon/V into the System SL, you can get the version numbers by running the compatibility mode version of Demon:

```
:run demonCM.pub.robelle,version
```

Examples of Calling Speed Demon

Copying the Examples

This chapter contains working examples of COBOL and Pascal source code that call Speed Demon. You can copy the examples from the manual, but typing them from scratch would be tedious and error-prone. The best way to copy the examples is to take them from the on-line Speed Demon documentation file. The file is stored in Robelle Qedit format. If you have Qedit, just Text a copy of Demon.Doc.Robelle and extract the portions that you are interested in.

If you don't have Qedit, use the Qcopy program to copy the documentation file to a new file in non-Qedit format, then use your favorite text editor to edit this file.

```
:run qcopy.qlib.robelle
>from=demon.doc.robelle;to=mydemon.source;new
>exit
```

COBOL Example

The following is an example COBOL program that shows how to call Speed Demon. This program prompts for a database, database password, dataset, and field list to process. Because upshifting is so messy in COBOL, you must enter the dataset name and field list in uppercase. If you want to use the @-field-list, just specify @ when asked by the sample program.

```
$control nolist
$control source,errors=10
 identification division.
 program-id.    testread.
date-written.  May 11, 1990.
 security.      Copyright Robelle Solutions Technology Inc.
 remarks.

*********************************************************
*                                                       *
*               testread - test spde intrinsics.        *
*                                                       *
*  version:  1.0                                        *
*  purpose:                                             *
*                                                       *
*  General-purpose program to test Speed Demon.         *
*                                                       *
*  Demonstrates that you must call DBOPEN before you call *
*  the Speed Demon procedures.                          *
*                                                       *
*********************************************************

 environment division.
 configuration section.
 source-computer.  hp-3000 series 37.
 object-computer.  hp-3000 series 37.
 special-names.
     top is new-page.

 input-output section.
 file-control.
     select line-printer assign to "LINEPRT".

 data division.
 file section.
 fd  line-printer
     data record is line-record.
 01  line-record            pic x(132).

$page "CONSTANTS"
 working-storage section.
 01  true-value             pic x value "T".
 01  false-value            pic x value "F".

$page "VARIABLES"
 01  line-count             pic s9(4) comp.
 01  page-no                pic s9(4) comp.

 01  input-line                  pic x(80).
     88  answer-spaces           value spaces.

 01  end-of-db-set-flag          pic x.
     88  end-of-db-set           value "T".

 01  in-record-count             pic s9(9) comp.

 01  display-number              pic -zzz,zz9.
$page "image area"

 01  image-area.
     05  db-all-list                pic x(2) value "@ ".
     05  db-same-list               pic x(2) value "* ".
     05  db-null-list               pic s9(4) comp value 0.
     05  db-dummy-arg               pic s9(4).
     05  db-password                pic x(8).
     05  db-mode0                   pic s9(4) comp value 0.
     05  db-mode1                   pic s9(4) comp value 1.
     05  db-mode2                   pic s9(4) comp value 2.
     05  db-get-serial  redefines db-mode2 pic s9(4) comp.
     05  db-mode3                   pic s9(4) comp value 3.
     05  db-rewind-set  redefines db-mode3 pic s9(4) comp.
```

```
     05  db-get-backwards redefines db-mode3 pic s9(4) comp.
     05  db-mode4                    pic s9(4) comp value 4.
     05  db-get-direct  redefines db-mode4 pic s9(4) comp.
     05  db-mode5                    pic s9(4) comp value 5.
     05  db-get-chained redefines db-mode5 pic s9(4) comp.
     05  db-mode6                    pic s9(4) comp value 6.
     05  db-get-previous redefines db-mode6 pic s9(4) comp.
     05  db-mode7                    pic s9(4) comp value 7.
     05  db-get-keyed   redefines db-mode7 pic s9(4) comp.
     05  db-status-area.
         10  db-cond-word            pic s9(4) comp.
             88  db-stat-ok              value zeros.
             88  db-end-of-chain         value 15.
             88  db-begin-of-chain       value 14.
             88  db-no-entry             value 17.
             88  db-end-file             value 11.
             88  db-begin-file           value 10.
         10  db-stat2                pic s9(4) comp.
         10  db-stat3-4              pic s9(9) comp.
         10  db-chain-length         pic s9(9) comp.
             88  db-empty-chain      value zeros.
         10  db-stat7-8              pic s9(9) comp.
         10  db-stat9-10             pic s9(9) comp.

$page "db- variables"

 01  db-base.
     05  filler                  pic x(2)  value spaces.
     05  db-name                 pic x(26).

 01  db-set                      pic x(16).

 01  db-list                     pic x(80).

 01  db-set-200                  pic s9(4) comp value 200.

 01  spde-control.
     05  spde-version            pic s9(4) comp value 0.
     05  spde-buffer-size        pic s9(9) comp value zeros.
     05  filler                  pic x(20) value spaces.

 01  db-buffer                   pic x(4096).

$page "[00] MAINLINE"
 procedure division.
 00-main                         section.

    perform 05-get-parameters
       thru 05-get-parameters-exit.

    if not answer-spaces then
        perform 10-init-testread
           thru 10-init-testread-exit
         move false-value        to end-of-db-set-flag
         move zeros              to in-record-count
         perform 20-read-set
            thru 20-read-set-exit
           until end-of-db-set
         move in-record-count to display-number
         display "In=", display-number.

    perform 90-close-files
       thru 90-close-files-exit.

 00-main-exit.  goback.
$page "[05] get-parameters"
*
* Prompt for the database, password, dataset and field list.
*
 05-get-parameters               section.
```

```
    perform 05-10-get-database.

    if not answer-spaces then
        perform 05-20-get-password
        if not answer-spaces then
            perform 05-30-get-dataset
            if not answer-spaces then
                perform 05-40-get-field-list.

    go to 05-get-parameters-exit.

05-10-get-database.
    move spaces                 to input-line.
    display "Enter Database Name".
    accept input-line.
    move input-line             to db-name.

05-20-get-password.
    move spaces                 to input-line.
    display "Enter Database Password".
    accept input-line.
    move input-line             to db-password.

05-30-get-dataset.
    move spaces                 to input-line.
    display "Enter Dataset Name (must be uppercase)".
    accept input-line.
    move input-line             to db-set.

05-40-get-field-list.
    move spaces                 to input-line.
    display "Enter @ or Field-List (must be uppercase)".
    accept input-line.
    move input-line             to db-list.

05-get-parameters-exit.  exit.
$page "[10] init-testread"
*
*  Open the database and initialize Speed Demon.
*

10-init-testread                section.

    perform 10-10-open-base.

    perform 10-20-init-spde.

    perform 10-30-display-suprinit.

    go to 10-init-testread-exit.

10-10-open-base.
    call "DBOPEN" using db-base
                        db-password
                        db-mode5
                        db-status-area.
    if not db-stat-ok then
        perform 99-fatal-error.

10-20-init-spde.
    call "SPDEDBINIT" using db-base
                            db-set
                            db-mode2
                            db-status-area
                            spde-control
                            db-list.
    if not db-stat-ok then
        perform 98-supr-error.

10-30-display-suprinit.
    move db-stat2               to display-number.
```

```
     display "Entry Length: ", display-number.
     move db-stat3-4             to display-number.
     display "Buffer Size:  ", display-number.

 10-init-testread-exit.  exit.
$page "[20] read-set"
*
*  Read all the records in the dataset.
*

 20-read-set                     section.

     perform 20-10-get-next.

     if not end-of-db-set then
         add 1 to in-record-count
         perform 20-20-display-next.

     go to 20-read-set-exit.

 20-10-get-next.
     call "SPDEDBSCAN" using db-base
                             db-status-area
                             db-buffer
                             db-dummy-arg.
     if db-end-file then
         move true-value               to end-of-db-set-flag
     else
     if not db-stat-ok then
         perform 98-supr-error.

 20-20-display-next.
*     move db-stat3-4             to display-number.
*     display "Record# ", display-number.

 20-read-set-exit.  exit.
$page "[90] close-files"
*
*  Cleanup by closing the dataset and the database.
*

 90-close-files                  section.

     perform 90-10-close-dataset.

     perform 90-20-close-base.

     go to 90-close-files-exit.

 90-10-close-dataset.
     call "SPDEDBSHUT" using db-base
                             db-set
                             db-mode2
                             db-status-area
                             db-dummy-arg.
     if not db-stat-ok then
         perform 98-supr-error.

 90-20-close-base.
     call "DBCLOSE" using db-base
                          db-dummy-arg
                          db-mode1
                          db-status-area.
     if not db-stat-ok then
         perform 99-fatal-error.

 90-close-files-exit.  exit.
$page "[98] supr-error"

 98-supr-error                   section.
```

```
     call "SPDEEXPLAIN" using db-status-area.

     stop run.

 98-supr-error-exit.  goback.
$page "[99] fatal-error"

 99-fatal-error                   section.

     call "DBEXPLAIN" using db-status-area.

     stop run.

 99-fatal-error-exit.  goback.
```

Pascal Example

Calling Speed Demon from Pascal is more difficult than from other languages due to Pascal's tighter type-checking on procedure parameters. The following is a complete example program written in Pascal/V that reads the `process` dataset of the `menu.qlib` database. This example program compiles and runs in native mode under MPE/iX, but you must make the two changes indicated. Because the buffer length parameter is not 32-bit aligned, Pascal/iX produces a warning when compiling the spde_db_control type. Note that the $check_actual_parm$ setting is left at 2 for the entire program. This is required to prevent parameter type mismatches between Speed Demon and Pascal.

```
$list off$
{ $HP3000_16$  <-- Require this compiler option for MPE/iX}
program driver(input,output);
type
   longint  = integer;
   shortint = -32768 ..  +32767;{Remove this line on MPE/iX}

   type_db_base =
      record
         db_id   : packed array[1..2] of char;
         db_name : packed array[1..14] of char;
      end;

   type_db_set =
      record
         name : packed array[1..16] of char;
      end;

   type_db_status_area = array[1..10] of shortint;

   type_db_mode = array[1..1] of shortint;

   type_db_buffer = array[1..46] of shortint;

   type_spde_db_control =
      record
         spde_version         : shortint;
         spde_buffer_size     : integer;
         spde_future          : packed array[1..20] of char;
      end;

   type_db_dummy_arg = array[1..2] of shortint;

var
   spde_db_control : type_spde_db_control;

   db_status_area : type_db_status_area;

   db_dummy_arg   : type_db_dummy_arg;

   db_mode1       : type_db_mode;
   db_mode2       : type_db_mode;

   db_base : type_db_base;

procedure dbopen; intrinsic;
procedure dbexplain; intrinsic;

function proctime : longint; intrinsic;
function timer : longint; intrinsic;

$check_formal_parm 2$
$check_actual_parm 2$
procedure spdedbinit(var db_base : type_db_base;
                     var db_set  : type_db_set;
                     var db_mode : type_db_mode;
                     var db_stat : type_db_status_area;
                     var control : type_spde_db_control;
                     var dummy   : type_db_dummy_arg
                    ); external spl;
procedure spdedbscan(var db_base   : type_db_base;
                     var db_status : type_db_status_area;
                     var db_buffer : type_db_buffer;
                     var dummy     : type_db_dummy_arg
                    ); external spl;
procedure spdedbshut(var db_base : type_db_base;
                     var db_set  : type_db_set;
                     var db_mode : type_db_mode;
                     var db_stat : type_db_status_area;
                     var dummy   : type_db_dummy_arg
                    ); external spl;
```

```
procedure spdeexplain(var status : type_db_status_area
                     ); external spl;
$check_formal_parm 3$

function open_base : boolean;
var
   db_password : packed array[1..8] of char;
   mode1       : shortint;
begin
   open_base := false;
   db_base.db_id    := '  ';
   db_base.db_name := 'menu.qlib;    ';
   db_password     := ';;      ';
   mode1           := 1;
   dbopen(db_base
         ,db_password
         ,mode1
         ,db_status_area
         );
   if db_status_area[1] <> 0 then
      dbexplain(db_status_area)
   else
      open_base := true;
end (*open_base*);

function init_spde : boolean;
var
  db_set : type_db_set;
begin
   init_spde := false;
   with spde_db_control do
      begin
         spde_version    := 0;
         spde_buffer_size:= 0;
         spde_future     := '                      ';
      end;
   db_set.name := 'PROCESS;        ';
   spdedbinit(db_base
             ,db_set
             ,db_mode1
             ,db_status_area
             ,spde_db_control
             ,db_dummy_arg
             );
   if db_status_area[1] <> 0 then
      spdeexplain(db_status_area)
   else
      init_spde := true;
end (*init_spde*);

function read_set : boolean;
var
   db_buffer : type_db_buffer;
begin
   read_set := false;
   spdedbscan(db_base
             ,db_status_area
             ,db_buffer
             ,db_dummy_arg
             );
   if db_status_area[1] = 0 then
      read_set := true
   else
   if db_status_area[1] <> 11 then
      spdeexplain(db_status_area);
end (*read_set*);

procedure test_spde;
var
   start_cpu : longint;
   start_wall: longint;
```

```
   in_count  : longint;
begin
   start_cpu := proctime;
   start_wall:= timer;
   in_count  := 0;
   while read_set do
      in_count := in_count + 1;
   start_cpu := proctime - start_cpu;
   start_wall:= timer - start_wall;
   writeln('In=',in_count:1);
   write('Cpu-time=',start_cpu:1,' ');
   write('Wall-time=',start_wall:1);
   writeln;
end (*test_spde*);

procedure complete_driver;
var
  db_set : type_db_set;
begin
   db_set.name := 'PROCESS;        ';
   spdedbshut(db_base
              ,db_set
              ,db_mode2
              ,db_status_area
              ,db_dummy_arg
              );
   if db_status_area[1] <> 0 then
      spdeexplain(db_status_area);
end (*complete_driver*);

begin (*driver*)

   db_mode1[1] := 1;
   db_mode2[1] := 2;
   if open_base then
      if init_spde then
      begin
         test_spde;
         complete_driver;
      end;
end.
```

Speed Demon Error Messages

Error Messages and Numbers

Speed Demon returns errors similar to IMAGE. Each error is associated with an error number. The following are the error numbers, descriptions, and causes that can be returned by any Speed Demon intrinsic.

-538: SPDEPREFETCH JCW VALUE n *IS LESS THAN ZERO OR GREATER THAN 5*

The SPDEPREFETCH JCW determines if Speed Demon should prefetch data into memory and if so, how far ahead it should read. A value of zero tells Speed Demon not to prefetch data. Values greater than five are not allowed. Check the value of the SPDEPREFETCH JCW and set it to an appropriate number.

-537: FINDJCW FAILED WITH ERROR# n

A call to the FINDJCW intrinsic failed. The value of *n* is the error number returned from FINDJCW. This error should never occur. If it does, please report it to Robelle Solutions Technology Inc.

-536: ROBELLE PREFETCH FAILURE ERROR# n

The internal Robelle prefetch module has returned an error. The value of *n* is the prefetch error number. This message is followed by a prefetch error message. You should report this error to Robelle.

-535: SPEED DEMON HAS EXPIRED

Your version of Speed Demon has expired. If you have received a nonexpiring version of Speed Demon/V, be sure to install it in the System SL. For Speed Demon/iX, make sure that you have restored DemonXL.Pub.Robelle. If you are using Speed Demon/iX from your own XL file, be sure to reinstall Speed Demon/iX after restoring the nonexpiring version.

-534: SPDEDBINIT NOT CALLED

Before calling SPDEDBSCAN or SPDEDBSHUT, you must call SPDEDBINIT successfully. Make sure that SPDEDBINIT was actually called and that the condition word returned from SPDEDBINIT was zero.

In Speed Demon/V, this error can also be caused by corrupting the stack. The Speed Demon intrinsics use two words of the DL-area (DB-32 and DB-14) to store global

information. Setting the first word (DB-32) to zero, possibly because of a bug, will cause this error.

-533: FLIMIT OF ACTUAL MPE FILE IS INVALID

SPDEDBINIT verifies that the physical MPE file that corresponds to the dataset matches the information returned by DBINFO. SPDEDBINIT computes the number of records in the MPE file as follows:

#records = (capacity - 1) / blockfactor + 1

where capacity and blockfactor are reported by the FORM command of Suprtool or QUERY. This number must match the EOF of the MPE file (you can check this with :listf,2).

This error is usually caused by restoring a earlier version of the dataset from a backup tape.

-532: INVALID BLOCK SIZE FOR DATASET: xx

The MPE file corresponding to the dataset has a block length that is less than the one reported from DBINFO. This can only happen if the MPE file is corrupted.

-531: INVALID BLOCKING FACTOR FOR DATASET: xx

The MPE file corresponding to the dataset has a blocking factor other than one. This can only happen if the MPE file is corrupted.

-530: INVALID FILECODE FOR DATASET: xx

This error should never occur. If it does, please report it to Robelle Solutions Technology Inc.

-529: SPEED DEMON NOT ALLOWED ON REMOTE DATABASES

Speed Demon only works with databases on the local machine. Check for a :File command specifying the database on a remote machine.

-528: SPEED DEMON REQUIRES FULL-READ ACCESS TO THE DATASET

The password used in DBOPEN must grant read access to the entire dataset passed to SPDEDBINIT. The creator password always grants full-read access to every dataset.

-527: THE DATASET CHANGED SINCE CALLING SPDEDBINIT

The dataset name or dataset number passed to SPDEDBINIT must be the same one passed to SPDEDBSHUT.

-526: INVALID EXTRA DATA SEGMENT IDENTIFIER

This error should never occur. If it does, please report it to Robelle Solutions Technology Inc.

-525: DATASET IS ALREADY ACTIVE

You can only use Speed Demon to read one dataset at a time. Two calls were made to SPDEDBINIT. Make sure that SPDEDBSHUT is called before calling SPDEDBINIT again.

-524: INVALID DATA SEGMENT INDEX

This error can only happen in Speed Demon/V. The DL-area of the stack has been corrupted. Make sure that DB-32 and DB-14 are not used by the calling program.

-523: UNABLE TO OBTAIN EXTRA DATA SEGMENT

Speed Demon needs an extra data segment the length of the buffer size plus a few hundred bytes (the default buffer size is 28,672 bytes). SPDEDBINIT was unable to allocate this extra data segment. Either the system is out of resources (e.g., DST table is full) or the maximum extra data segment size is configured too small.

-522: INVALID FUTURE PARAMETER

The future part of the control record does not contain twenty blanks. Make sure that the control record was declared correctly, and that the correct name was passed to SPDEDBINIT. Verify that the control record is the fifth parameter of SPDEDBINIT, and be sure that either a dummy argument or a field list was passed as the last parameter to SPDEDBINIT.

-521: INVALID BUFFER LENGTH: xxxx

The buffer length of the control record did not contain a valid value. The minimum buffer size is 4096 and the maximum is 14,336, unless zero was specified. Make sure that the control record was declared correctly, the correct name was passed to SPDEDBINIT, verify that the control record is the fourth parameter of SPDEDBINIT, and be sure that a dummy argument was passed as the last parameter to SPDEDBINIT.

-520: INVALID VERSION NUMBER: x

The version number of the control record did not contain a zero. Make sure that the control record was declared correctly, the correct name was passed to SPDEDBINIT, verify that the control record is the fifth parameter of SPDEDBINIT, and be sure that a dummy argument was passed as the last parameter to SPDEDBINIT.

Assertion Errors

It is possible for Speed Demon to abort because of an internal inconsistency. These are called assertion errors and they should be reported to Robelle Solutions Technology Inc. There is one assertion error that can be caused by problems in the application software.

Each Speed Demon/V intrinsic checks for ten words of room in the status area. If there is not enough room, Speed Demon/V is aborted with assertion error number one. If this happens, you should check that the declaration of the status area is correct. This problem could also be caused by a large portion of the application stack being wiped out.

Welcome to Calling Suprtool

Calling Suprtool

Suprtool, including its Suprlink and Dbedit components, is a utility program, like FCOPY or QUERY. You run it, either interactively or in a batch job, and feed it commands to define a task to be done. How would a user application program invoke Suprtool to perform a desired task? One way would be to use the COMMAND intrinsic to launch a batch job that ran Suprtool. Unfortunately, the user program would have little control over when the batch job started or finished.

To solve this problem, Robelle provides an interface routine that will run Suprtool for a user program, and pass commands from the program to Suprtool (the same commands you would type into Suprtool). This routine (procedure, subroutine, intrinsic) allows user programs to "call" Suprtool. A typical use of this interface would be for a COBOL program to ask Suprtool to extract a selected subset from a large IMAGE dataset and write it to a disc file, which the COBOL program would then read and format into a report.

Suprtool2 Routine

The interface routine is called Suprtool2 (not Suprtool). User programs written in COBOL, COBOL/iX, FORTRAN, FORTRAN/iX, Pascal, Pascal/iX, SPEEDWARE, SPL, or TRANSACT can call Suprtool2 and ask Suprtool to do any of the normal Suprtool tasks such as copy, extract, or sort. The routine creates Suprtool as a son process.

The user program instructs Suprtool by calling the Suprtool2 routine repeatedly with Suprtool command lines. When the first Suprtool command is sent, the interface builds temporary files which will be used for input and output to Suprtool. When the user program sends an Exit command in a separate call to the interface, the interface creates Suprtool as a son process. Finally, the interface prints the $stdlist message file, if so directed by the user program.

Importance of the Exit Command

The interface will not invoke Suprtool until your program passes `Exit` to it as a command line. The Exit command must be alone and left-justified in the command line. You may use Xeq to separate multiple tasks, but none of the tasks will be executed until you pass Exit to the interface. If you forget the final Exit or put it in the same command line with another command such as Xeq, your Suprtool tasks will be ignored.

Control Record

The user program must pass a special control record to the interface on each call. For COBOL, we suggest that this control record be placed in the COPYLIB and copied into programs that require the Suprtool interface. The most common error in using the Suprtool2 interface is typing the control record incorrectly. Use the file Cobol.Qlibsrc.Robelle instead. You can copy the control record directly into your COBOL program with the following Qedit command (use /JOIN with EDIT/3000):

```
/add 50.1=cobol.qlibsrc.robelle
```

The definition of the control record, with the proper initializing values, is as follows:

```
01  supr-control.
    05  supr-version              pic s9(4) comp value 4.
    05  supr-status               pic s9(4) comp.
        88  supr-ok               value zeros.
        88  supr-bad-msgfiles          value 1.
        88  supr-aborted               value 2.
        88  supr-create-error          value 3.
        88  supr-bad-total-type        value 4.
    05  supr-command-line         pic x(256) value spaces.
    05  supr-flags.
        10  supr-priority         pic x(2) value spaces.
            88  supr-priority-cs       value "CS".
            88  supr-priority-ds       value "DS".
            88  supr-priority-es       value "ES".
        10  supr-maxdata          pic s9(9) comp value 0.
        10  supr-print-state      pic x(2) value "ER".
            88  supr-print-on-error    value "ER".
            88  supr-print-always      value "AL".
            88  supr-print-never       value "NE".
        10  supr-total-type       pic x(2) value "CO".
            88  supr-total-cobol  value "CO".
            88  supr-total-ascii  value "AS".
        10  supr-other-flags      pic x(18) value spaces.
    05  supr-totals pic s9(17) sign is trailing
                separate character occurs 15 times.
    05  supr-out-count            pic s9(9) comp.
    05  supr-workspace            pic x(20) value spaces.
```

Status

The supr-status field returns a 0 if the command line was sent to Suprtool without incident or one of the error numbers shown as 88 levels.

Command Line

The supr-command-line can contain any Suprtool command. Use the same format that you use in typing commands into Suprtool. You don't need to enter commands as a single string of 256 characters in a single call to the interface. You may use ";" to send several commands in one string, or you may use the "&" mechanism to continue commands.

The final call must have Exit as the command, alone and left-justified in the command line. The final Exit command can be in uppercase or lowercase, but cannot be abbreviated. MPE commands can be passed into the interface and Suprtool will execute them.

Priority

The supr-priority flag should contain one of "CS", "DS", or "ES". The default is "DS".

Maxdata

The supr-maxdata contains the Maxdata in words for Suprtool. The default is 32,000, but you can ask for a smaller stack using this parameter (e.g., 20,000). There is no problem with reducing Maxdata when you plan to Sort. However, if you plan to do copies without sorts, you should send Suprtool a Buffer 4096 command if you reduce Maxdata. Otherwise, you are likely to get a Stack Overflow abort within Suprtool or the error message "Unable to allocate buffer in DL area". If you want the default, this field should contain a 0.

Print State

If the supr-print-state contains "AL", the output from Suprtool will always be printed on $stdlist. If the state is "NE", the output will never be printed. If the state is "ER" or blank, the output will be printed only if Suprtool aborts due to an error.

Total Type

The supr-total-type determines the format of the supr-totals array. If you call Suprtool2 from COBOL, you should use "CO". The COBOL format is display (with leading zeros) and a trailing sign. If the type is "AS", each total is returned left-justified in the total field with a leading sign.

Totals

If you specify the Total command as part of an extract task, Suprtool2 returns the totals in the supr-totals array. Totals are returned in exactly the same order in which they were specified. If you are calling Suprtool2 from COBOL, never specify the decimal-precision portion of the Total command. If your total includes an implied decimal point, you will have to modify the supr-totals declaration to include an implied decimal point (e.g., pic s9(15)v99 ...).

If you specify "AS" as the supr-total-type, each total is formatted as an 18-byte string. In this case, you should specify the correct decimal-precision in the Total command. The exponent portion of real totals is truncated by the Suprtool2 interface.

Out Count

After a successful call to the Suprtool2 interface, the supr-out-count is set to the number of Suprtool output records. The supr-out-count is only returned after the Suprtool2 call with the Exit command. Suprtool also puts this count in the SuprtoolOutCount JCW, up to the maximum JCW value permitted, 65,535.

Workspace

The supr-workspace part of the record MUST contain spaces before the first call to the Suprtool2 procedure.

Compiling and Linking on MPE V

You require PH capability in order to use the Suprtool2 interface. The classic object code for the interface resides in ST2USL.Pub.Robelle. If your program aborts with a "Stack Overflow" you should increase the size of Maxdata. Users who call the interface from FORTRAN programs will have to increase the Maxdata size. COBOL programmers should wait until they get a "Stack Overflow".

The interface is installed in SL files. The following commands demonstrate how to compile and prepare your program to use the Suprtool interface. This example assumes that the Suprtool2 routine has been installed in the System SL:

```
:cobol example.source
:prep $oldpass,example.pub;cap=ph
:save example.pub
:run example.pub
```

Using Lib=

The installation chapter of the manual tells how to install the interface so that it is available to every program. You can also install the interface routine in a SL file and run your program with Lib=. Here is an example of how to add the Suprtool segment from ST2USL.Pub to your existing SL file.

```
:run fcopy.pub.sys
>from=st2usl.pub.robelle;to=st2usl.pub;new
>exit
:segmenter
-buildsl sl.pub,400,4    {fails if you already have an SL}
-sl sl.pub
-usl st2usl.pub
-addsl suprtool
-exit

:cobol tool2cob.source
:prep $oldpass,tool2cob.program;cap=ph
:save tool2cob.program

:run tool2cob.program;lib=p
```

The MPE :prep command prints a warning message when you specify cap=ph. You may ignore this message or specify cap=ia,ba,ph on your :prep command. If your program requires other capabilities, they should also be specified (e.g., cap=ph,ds).

Warning: The Suprtool2 interface will change over time. The interface should never be installed directly into a program file. It should only be installed into SL files.

Compiling and Linking on MPE/iX

You require PH capability in order to use the Suprtool2 interface. The interface is installed in XL files. The following commands demonstrate how to compile and link your program to use the native mode Suprtool interface.

```
:cob85xl example.source
:link from=$oldpass;to=example.pub;cap=ph
:run example.pub;xl='st2xl.pub.robelle'
```

User XL Files

You can use MPE/iX's Linkedit command to add the Suprtool2 interface to your own XL file. For example,

```
:linkedit
-xl xl.pub
-copyxl from=st2xl.pub.robelle;replace
-exit
:run example.pub;xl='xl.pub'
```

The Replace option is not in all versions of Linkedit. If your version does not allow Replace, and you already have Suprtool2 in your XL file, you will need to manually purge the existing module before copying the new one:

```
:linkedit
-xl xl.pub
-purgexl module=suprtool.asm
```

Suprtool2 can successfully be installed into the System XL file, but this is not recommended by HP. Suprtool2 can be combined into an XL file with other Robelle XL files, except Qcompxl (a part of Qedit).

Suprtool Run Parameters

By default, the Suprtool2 interface runs Suprtool as Suprtool.Pub.Robelle with Lib=S. We have provided JCWs that allow Suprtool to be moved or run with a different library.

Moving Suprtool

To change the location of the Suprtool program file, you must specify a JCW and use a :File command for the program file:

```
:setjcw suprtool2filecommand = 2
:file suprtool = suprtool.pub.dev
:run example.pub;lib=p
```

We suggest you add the Setjcw and :File commands to a logon UDC. This way, every invocation of the Suprtool2 interface will use the copy of Suprtool in the Pub group of the Dev account.

Previous versions of the Suprtool2 interface accepted a value of 1 for the suprtool2filecommand JCW. In this case, you must specify a file command for Suprtool.Pub.Robelle. We don't recommend you do this, since this only works on MPE/iX and not on MPE V.

Lib=

The Suprtool2 interface looks at the Suprtool2lib JCW for which library to use. The values of this JCW correspond to the description of the library portion of the flag variable of the create intrinsic. These values are:

JCW Value	uses lib
0	Lib=S
1	Lib=P
2	Lib=G
3	<invalid>

If the Suprtool2lib JCW does not exist, the Suprtool2 interface runs Suprtool with Lib=S. To have the Suprtool2 interface always invoke Suprtool with Lib=P, you would use this command:

```
:setjcw suprtool2lib = 1
```

Slist234 File

Some previous versions of the Suprtool2 interface created a temporary file called Slist234, which contained the $stdlist from the Suprtool run. A few programs were written to read this file. By default, the Suprtool2 interface no longer creates this file. If your program needs it, use the following JCW to force the interface to create and save the Suprtool listing in the temporary Slist234 file:

```
:setjcw saveslist234 = 1
```

Lockwords on Suprtool

To tell the Suprtool2 interface that the Suprtool program has a lockword you can use the suprtoolfilecommand facility.

```
file suprtool=suprtool/secret.pub.robelle
setjcw suprtool2filecommand = 2
run prog;lib=p
```

Examples of Calling Suprtool

Copying the Examples

This chapter contains working examples of source code that calls Suprtool2. You can copy the examples from the manual, but typing them from scratch would be tedious and error-prone. The best way to copy the examples is to take them from the on-line Suprtool2 documentation file. The file is stored in Robelle Qedit format. If you have Qedit, just Text a copy of Suprcall.Doc.Robelle and extract the portions that you are interested in.

If you don't have Qedit, use the Qcopy program to copy the documentation file to a new file in non-Qedit format, then use your favorite text editor to edit this file.

```
:run qcopy.qlib.robelle
>from=suprcall.doc.robelle;to=mysupr2.source;new
>exit
```

COBOL Example

Below is a sample COBOL program named TOOL2COB. It calls the Suprtool2 interface procedure. The purpose of TOOL2COB is to print selected item master entries from an inventory database, sorted by item number. The program uses Suprtool to create a disc file named SELITEM, filled with the selected item numbers and their descriptions. Then it reads the disc file and prints a report on the line printer.

The database used in this example is called INVORY, and the dataset is called INVREC. Here is a QUERY FORM listing of the dataset:

```
:run query.pub.sys
>base=invory.data
PASSWORD = >>
MODE = >>1
>form invrec

SET NAME:
   INVREC,DETAIL

      ITEMS:
         ITEM,                 X8       <<SEARCH ITEM>>
         DESCRIPT,             X50
         WEIGHT,               J2
         STD-PKG,              X12
         SELLPRCE,             J2
         SPUOM,                X2
         SPLEVEL,              3J2
         SPMULTIR,             3J1
              CPMULTIR,             J1
         INDIC,                X2
         VENDORPR,             X6
         SUPPNUM,              X2
         PRCEDATE,             X6

      CAPACITY: 44200           ENTRIES: 35659
```

The listing for the COBOL program is:

```
$control source,uslinit,errors=15,list
$control debug,bounds
 identification division.
 program-id.    tool2cob.
 author.        Robert M. Green.
 date-written.  19 apr83.
 date-compiled.
 installation.  Robelle Solutions Technology Inc.
 security.      Copyright 1983-2001, Robelle Solutions Technology Inc.
 remarks.

*************************************************************
*                                                           *
*                       TOOL2COB                            *
*                                                           *
*  FUNCTIONAL OVERVIEW:  uses Suprtool to extract and print*
*                                                           *
*  INPUT:  Invrec set of INVORY database.                   *
*          All items with spec prefix (4 char), sorted.     *
*                                                           *
*  OUTPUT: print report on LINE-PRINTER                     *
*                                                           *
*  TERMINATED BY:  eof or error                             *
*                                                           *
*  PREP WITH:      cap = ph                                 *
*                                                           *
*  REV  ,INIT,DATE    , REASON:                             *
*  00,  rmg, 19 Apr83, started first implementation.        *
*  01,  djg, 25 Jan84, modified the control record to       *
*                      include the version number.          *
*  02,  djg, 09 Nov87, modified to include totals in the    *
*                      control record.                      *
*  03,  djg, 10 Aug90, modified for version 4 control rec.  *
*                                                           *
*************************************************************

$page "environment division"
 environment division.
 configuration section.
 source-computer. hp-3000.
 object-computer. hp-3000.
 special-names.
top is new-page.

 input-output section.
 file-control.
select line-printer assign to "LINEPRT,UR,A,LP".
select selitem assign to "SELITEM".

 data division.
 file section.
 fd  selitem
data record is item-record.
 01  item-record.
     05  item-number.
         10  item-prefix         pic x(4).
         10  item-suffix         pic x(4).
     05  item-description        pic x(50).
 fd  line-printer
     data record is line-record.
 01  line-record                 pic x(132).

$page "constants"
 working-storage section.
 01  true-value                  pic x  value "T".
 01  false-value                 pic x  value "F".
 01  revision-no                 pic 99 value 03.

$page "logical variables"
 01  end-of-items-flag           pic x.
     88  end-of-items            value "T".
```

```
$page "variables"
 01  if-command.
     05  filler                  pic x(10) value
                "IF  ITEM='".
     05  sel-prefix              pic x(4).
     05  filler                  pic x(2) value "' ".

$page "input area"
 01  accept-buffer.
     05  input-buf               pic x(80).
         88  answer-spaces       value spaces.

$page "report layouts"
 01  report-record.
     05  filler                  pic x(5) value spaces.
     05  report-prefix           pic x(4).
     05  filler                  pic x value "-".
     05  report-suffix           pic x(4).
     05  filler                  pic x(3) value spaces.
     05  report-description      pic x(50).

$page "Suprtool Control Parameter"

* suprtool control area
 01  supr-control.
     05  supr-version              pic s9(4) comp value 4.
     05  supr-status               pic s9(4) comp.
         88  supr-ok               value zeros.
         88  supr-bad-msgfiles          value 1.
         88  supr-aborted               value 2.
         88  supr-create-error          value 3.
         88  supr-bad-total-type        value 4.
     05  supr-command-line         pic x(256) value spaces.
     05  supr-flags.
         10  supr-priority         pic x(2) value spaces.
             88  supr-priority-cs       value "CS".
             88  supr-priority-ds       value "DS".
             88  supr-priority-es       value "ES".
         10  supr-maxdata          pic s9(9) comp value 0.
         10  supr-print-state      pic x(2) value "ER".
             88  supr-print-on-error    value "ER".
             88  supr-print-always      value "AL".
             88  supr-print-never       value "NE".
         10  supr-total-type       pic x(2) value "CO".
             88  supr-total-cobol  value "CO".
             88  supr-total-ascii  value "AS".
         10  supr-other-flags      pic x(18) value spaces.
     05  supr-totals pic s9(17) sign is trailing
                 separate character occurs 15 times.
     05  supr-out-count            pic s9(9) comp.
     05  supr-workspace            pic x(20) value spaces.

$page "procedure division"

************************************************************
*                                                          *
*           P R O C E D U R E   D I V I S I O N            *
*                                                          *
************************************************************

 procedure division.
 00-main                          section.

* Ask for selection criteria.

     display "TOOL2COB ", revision-no.
     display " ".

     display "Enter 4-character item prefix to select:".
```

```
     move spaces to input-buf.
     accept input-buf.
     if answer-spaces then go to 00-main-exit.

     move input-buf to sel-prefix.

* Use suprtool to build the extract file.

     move "AL" to supr-print-state.

     move "base invory.data,5,dev" to supr-command-line.
                                   perform 01-call-suprtool.
     move "get invrec"             to supr-command-line.
                                   perform 01-call-suprtool.
     move if-command               to supr-command-line.
                                   perform 01-call-suprtool.
     move ":purge selitem"         to supr-command-line.
                                   perform 01-call-suprtool.
     move "output selitem"         to supr-command-line.
                                   perform 01-call-suprtool.
     move "extract item,descript" to supr-command-line.
                                   perform 01-call-suprtool.
     move "sort item"              to supr-command-line.
                                   perform 01-call-suprtool.
     move "exit"                   to supr-command-line.
                                   perform 01-call-suprtool.

* At this point, the file "SELITEM" contains the sorted
* data. This file must be read and printed on the line-
* printer.

     open output line-printer.
     open input selitem.
     move false-value to end-of-items-flag.

     perform 02-print-item
       until end-of-items.

     close selitem.
     close line-printer.

 00-main-exit. goback.

 01-call-suprtool.
     call "SUPRTOOL2" using supr-control.
     if not supr-ok then
         display "Error:  Unable to sort the invrec dataset"
         display " "
         display "Suprtool interface error number: ",
                 supr-status.

 02-print-item.

     read selitem at end
       move true-value to end-of-items-flag.

     if not end-of-items then
       move item-prefix       to report-prefix
       move item-suffix       to report-suffix
       move item-description  to report-description
       move report-record to line-record
       write line-record after advancing 1 lines.
```

FORTRAN Example

FORTRAN programmers may use the Suprtool2 interface to invoke Suprtool from within a FORTRAN program. The control area may be in COMMON or it may be local to one routine. If the control area is declared local to one routine, then all calls

to Suprtool2 must be from within the same routine. The following example declares the control area in a common block. One routine initializes the Suprtool2 control area. The second routine is repeatedly passed command lines which copy the first 100 lines of catalog.pub.sys to the temporary file "tempfile". This example program is compatible with FORTRAN 66, FORTRAN 77/V, and FORTRAN 77/iX.

FORTRAN 66

FORTRAN 66 users may need to increase the size of Maxdata for your program in order to use the interface. The $HP3000_16 ALIGNMENT$ option is not needed by FORTRAN 66 and if it is not deleted a warning is produced by the compiler. FORTRAN 66 produces a warning on the line `call setjcw`, but this warning may be safely ignored.

FORTRAN 77/V and FORTRAN 77/iX

The example program is valid for FORTRAN 77/V and FORTRAN 77/iX, but you must use the $HP3000_16 ALIGNMENT$ compiler option with FORTRAN 77/iX. The $HP3000_16 ALIGNMENT$ compiler option is required for FORTRAN 77/iX, but the option is not needed for FORTRAN 77/V. If the option is not deleted a warning is produced by the FORTRAN 77/V compiler. In FORTRAN 77, the `suprcommand` variable may be declared as `character*256` eliminating the need for the `suprfiller`.

```
$HP3000_16 ALIGNMENT$
      program test
      character*254 command
      call initsuprcontrol
      command = "input catalog.pub.sys"
      call invokesuprtool(command)
      command = "numrecs 100"
      call invokesuprtool(command)
      command = "output tempfile,temp"
      call invokesuprtool(command)
      command = "exit"
      call invokesuprtool(command)
20    stop
      end
C  This subroutine initializes the control record for SUPRTOOL.
C  It should be called once at the beginning of the program.
C
      subroutine initsuprcontrol
      common /suprtool/ suprcontrol
      integer*2 suprcontrol(291)
C
      integer*2     suprversion
      integer*2     suprstatus
      character*254 suprcommand
      character*2   suprfiller
      character*2   suprpriority
      integer*4     suprmaxdata
      character*2   suprprintstate
      character*18  suprotherflags
      character*2   suprtotaltype
      character*18  suprtotals(15)
      integer*4     suproutcount
      character*20  suprworkspace
C
      equivalence(suprversion   ,suprcontrol(1))
      equivalence(suprstatus    ,suprcontrol(2))
      equivalence(suprcommand   ,suprcontrol(3))
      equivalence(suprfiller    ,suprcontrol(130))
      equivalence(suprpriority  ,suprcontrol(131))
      equivalence(suprmaxdata   ,suprcontrol(132))
      equivalence(suprprintstate,suprcontrol(134))
      equivalence(suprtotaltype ,suprcontrol(135))
      equivalence(suprotherflags,suprcontrol(136))
      equivalence(suprtotals    ,suprcontrol(145))
      equivalence(suproutcount  ,suprcontrol(280))
      equivalence(suprworkspace ,suprcontrol(282))
C
      suprversion   = 4
      suprstatus    = 0
      suprcommand   = " "
      suprfiller    = "  "
      suprpriority  = "  "
      suprmaxdata   = 0
      suprprintstate= "ER"
      suprtotaltype = "AS"
      suprworkspace = " "
C
      return
      end
C  Call the "invokesuprtool" subroutine for each command line
C  that must be passed to SUPRTOOL.
C
      subroutine invokesuprtool(command)
      character*254 command
      common /suprtool/ suprcontrol
      integer*2 suprcontrol(291)
C
      integer*2     suprversion
      integer*2     suprstatus
      character*254 suprcommand
      character*2   suprfiller
```

```
      character*2   suprpriority
      integer*4     suprmaxdata
      character*2   suprprintstate
      character*18  suprotherflags
      character*2   suprtotaltype
      character*18  suprtotals(15)
      integer*4     suproutcount
      character*20  suprworkspace
C
      equivalence(suprversion   ,suprcontrol(1))
      equivalence(suprstatus    ,suprcontrol(2))
      equivalence(suprcommand   ,suprcontrol(3))
      equivalence(suprfiller    ,suprcontrol(130))
      equivalence(suprpriority  ,suprcontrol(131))
      equivalence(suprmaxdata   ,suprcontrol(132))
      equivalence(suprprintstate,suprcontrol(134))
      equivalence(suprtotaltype ,suprcontrol(135))
      equivalence(suprotherflags,suprcontrol(136))
      equivalence(suprtotals    ,suprcontrol(145))
      equivalence(suproutcount  ,suprcontrol(280))
      equivalence(suprworkspace ,suprcontrol(282))
C
      system intrinsic setjcw
C
      if(suprstatus.ne.0) goto 900
      suprcommand = command
      call suprtool2(suprcontrol)
      if (suprstatus.eq.0) goto 900
C     display "SUPRTOOL interface failed with command:"
C     display command
      call setjcw(-1)
900   return
      end
```

TRANSACT Example

To call the Suprtool2 interface from a TRANSACT program, you must first define the Suprtool2 control record. We suggest that the control record be included from the file Transact.Qlibsrc.Robelle. The TRANSACT definition of the control record, with the proper initializing values, is as follows:

```
define(item)
 supr-control          x(582),init = " ":
   supr-version        i(4,0,2)    = supr-control(1):
   supr-status         i(4,0,2)    = supr-control(3):
   supr-command        x(256)      = supr-control(5):
   supr-pri            x(2)        = supr-control(261):
   supr-maxdata        i(9,0,4)    = supr-control(263):
   supr-print          x(2)        = supr-control(267):
   supr-tot-type       x(2)        = supr-control(269):
   supr-other          x(18)       = supr-control(271):
   supr-totals      15 x(18)       = supr-control(289):
supr-total         9(17)       = supr-totals(1):
supr-tot-sign      x(1)        = supr-totals(18):
   supr-out-count      i(9,0,4)    = supr-control(559):
   supr-workspace      x(20)       = supr-control(563);

 list supr-control;

 << set up initial values >>

 let  (supr-version)  = 4;   << don't change >>
 let  (supr-status)   = 0;   << check after each call >>
 move (supr-command)  = " ";
 move (supr-pri)      = " "; << "CS","DS","ES"  >>
                             << "DS" is default >>
 let  (supr-maxdata)  = 0;
 move (supr-print)    = "ER";<< print on error only >>
                        << "NE" - never,       >>
                        << "AL" - always       >>
 move (supr-tot-type) = "CO";<< total type    >>
                        << "CO" - cobol, >>
                        << "AS" - ascii  >>
 move (supr-other)    = " ";
 move (supr-out-count)= 0;
 move (supr-workspace)= " ";
```

Below is a sample program that uses the TRANSACT include file to call the Suprtool2 interface. The program uses Suprtool to find two totals and the output count of selected records from the sales-hist dataset of the SH database. Be careful not to create commands greater than 256 characters in length.

```
SYSTEM SUPREX;
SET(OPTION) NOHEAD;
DEFINE(ITEM)
       TOTAL1      I (9,0,4),  EDIT = "ZZZZZZZZ^":
       TOTAL2      I (9,0,4),  EDIT = "ZZZZZZZZ^":
       CUSTNO      X (6),      INIT = "000000";

LIST   TOTAL1,
       TOTAL2,
       CUSTNO;

!INCLUDE(TRANSACT.QLIBSRC.ROBELLE)

DATA(SET)  CUSTNO;

MOVE (SUPR-COMMAND)           = "BASE SH,5,READ";
PERFORM CALL-SUPRTOOL2;

MOVE (SUPR-COMMAND)           = "GET SALES-HIST";
PERFORM CALL-SUPRTOOL2;

MOVE (SUPR-COMMAND)           = "DEF CUST-NO,HIST-KEY[1],6";
PERFORM CALL-SUPRTOOL2;

MOVE (SUPR-COMMAND)           = "IF CUST-NO = '" +
                                (CUSTNO) + "'";
PERFORM CALL-SUPRTOOL2;

MOVE (SUPR-COMMAND)           = "TOTAL CU-PER1-QTY";
PERFORM CALL-SUPRTOOL2;

MOVE (SUPR-COMMAND)           = "TOTAL CU-PER2-QTY";
PERFORM CALL-SUPRTOOL2;

MOVE (SUPR-COMMAND)           = "OUTPUT $NULL";
PERFORM CALL-SUPRTOOL2;

MOVE (SUPR-COMMAND)           = "EXIT";
PERFORM CALL-SUPRTOOL2;

IF (SUPR-TOT-SIGN(1)) = "-" THEN
   LET (TOTAL1)               = (SUPR-TOTAL(1)) * -1
ELSE
   LET (TOTAL1)               = (SUPR-TOTAL(1));

IF (SUPR-TOT-SIGN(2)) = "-" THEN
   LET (TOTAL2)               = (SUPR-TOTAL(2)) * -1
ELSE
   LET (TOTAL2)               = (SUPR-TOTAL(2));

DISPLAY
  "There are ": SUPR-OUT-COUNT, EDIT="ZZZZZZZZ^":
  "records for customer ": CUSTNO:
  "Period 1 Qty this year", COL=5, LINE: TOTAL1:
  "Last year ": TOTAL2;

EXIT;

CALL-SUPRTOOL2:

  PROC SUPRTOOL2((SUPR-CONTROL));
  IF (SUPR-STATUS) <> 0 THEN
     DISPLAY "SUPRTOOL2 INTERFACE ERROR  ", LINE = 2:
          SUPR-STATUS, JOIN = 1:
        "ON COMMAND:",LINE:
         SUPR-COMMAND;
  RETURN;

END SUPREX;
```

SPL Example

This SPL program invokes Suprtool through the interface routine:

```
$control errors=5,main=tool2spl
begin
<<
Copyright 1982-2001 Robelle Solutions Technology Inc.

    name:        tool2spl
    purpose:     test the suprtool interface.
    prep:        cap = ph
    run:         run with parm=2 for print stdlist always.
                 run with parm=3 to print stdlist never.
>>
define current'version = "(Version 0.6)"#;
$page "Global variables of the tool2spl program"

  << suprtool2 interface layout >>

  equate
     wl'supr'control      = 290
    ,wl'supr'command'line = 128
    ,wl'supr'totals       = 9
    ,bl'supr'totals       = wl'supr'totals * 2
    ,max'supr'totals      = 15
    ;

  integer array supr'control(0:wl'supr'control);
  integer array supr'version(*)      = supr'control(000);
  integer array supr'status(*)       = supr'control(001);
  integer array supr'command'line(*) = supr'control(002);
     byte array b'command'line(*)      = supr'command'line;
  integer array supr'flags(*)        = supr'control(130);
     byte array supr'priority(*)       = supr'flags(0);
   double array supr'maxdata(*)        = supr'flags(1);
     byte array supr'print'state(*)    = supr'flags(3);
     byte array supr'total'type(*)     = supr'flags(4);
  integer array supr'totals(*)       = supr'control(144);
   double array supr'out'count(*)    = supr'control(279);
  integer array supr'workspace(*)    = supr'control(281);

  equate supr'ok          = 0
        ,bad'msgfiles     = 1
        ,suprtool'aborted = 2
        ,create'error     = 3
        ;

  << standard global variables for Robelle SPL programs >>

integer
   copy'of'parm     << copy of "ACTUAL'PARM" >>
  ,actual'parm=q-4  << run ;parm=nn (use in mainline) >>
  ,input'length     << used only with input defines >>
  ;
logical
   batch            << run in batch mode >>
  ;
equate
   rtn = 13  << carriage return in ascii >>
  ,bell = 7  << ring the bell    in ascii >>
  ,bl'inbuf = 86
  ,bl'outbuf = 132
  ,wl'outbuf = bl'outbuf/2
  ,wl'inbuf = bl'inbuf/2
  ;
define
   end'if   = end#  ,end'else = end#  ,end'while= end#
  ,end'case = end#  ,end'do   = end#  ,end'proc = end#
  ,end'subr = end#
  ,p = begin move outbuf := #
  ,output=,2;output'(*);end#,err=,2;err'(*);end#
  ,ask=,2;ask'(*);end#, warn =,2;warn'(*);end#
  ,trim = while input'length>0 and
           inbuf'(input'length-1)=" " do
```

```
      input'length:=input'length-1#
  ,echo = if batch then
     begin trim; print(inbuf,-input'length,0); end#
  ,input = begin input'length := readx(inbuf,-bl'inbuf);
                 if > then quit(10); echo;
                 inbuf'(input'length):= rtn;
           end#
  ;

integer array inbuf (0:wl'inbuf);
  byte array inbuf'(*)=inbuf;

integer array outbuf  (0:wl'outbuf);
  byte array outbuf'(*) = outbuf;

intrinsic
   ascii, aritrap, binary, calendar, clock, dascii,
   dateline, dbinary, debug, genmessage, print, quit, readx,
   terminate, who, xaritrap, xcontrap, xlibtrap, xsystrap,
   setjcw;

  << external definition of suprtool2 routine >>
    procedure suprtool2(supr'definition);
       array supr'definition;
       option external, check 3;

$page "Standard Robelle Utility Procedures"

procedure output'( address ) ;
  value address; integer address; << in outbuf >>
begin   integer output'length;
  << check length for overflow of buf >>
  output'length := (address-@outbuf)*2;
  if output'length > bl'outbuf then begin
    print(outbuf,-bl'outbuf,0);
    quit(111);
    end'if;
  print(outbuf,-output'length,0);
  end'proc; <<output'>>

procedure ask' (address);
  value address; integer address;
begin   integer output'length;
  << check length for overflow of buf >>
  output'length := (address-@outbuf)*2;
  if output'length > bl'outbuf then begin
    print(outbuf,-bl'outbuf,0);
    quit(111);
    end'if;
  print(outbuf,-output'length,%320);
  end'proc; <<ask'>>

procedure err'(address);
  value address; integer address; << in outbuf >>
begin   integer output'length;
  integer array errbuf(0:10);
  move errbuf:=([8/bell ,8/10]<<lf>>,"ERROR:  ");
  print(errbuf,5,%320);
  << check length for overflow of buf >>
  output'length := (address-@outbuf)*2;
  if output'length > bl'outbuf then begin
    print(outbuf,-bl'outbuf,0);
    quit(111);
    end'if;
  print(outbuf,-output'length,0);
  print(errbuf,0,0);
  end'proc; <<err'>>

procedure blank(buf,count);
  value count;
  integer count;
```

```
  integer array buf;
begin
  if count > 0 then begin
    buf:="  ";
    if (count:=count-1)>0 then
      move buf(1):=buf,(count);
    end'if;
  end'proc; <<blank>>

procedure b'blank(buf,count);
  value count;
  integer count;
  byte array buf;
begin
  if count > 0 then begin
    buf:=" ";
    if (count:=count-1)>0 then
      move buf(1):=buf,(count);
    end'if;
  end'proc; <<b'blank>>

$page "level 1:  print'totals"
<<  Print the Suprtool2 totals.
>>

procedure print'totals;
begin
   byte array
      blank'total(0:bl'supr'totals)
;
   byte pointer
      supr'totals'
;
   integer
      total'index
     ,total'offset
;
   logical
      first'time
;

   b'blank(blank'total,bl'supr'totals);
   first'time  := true;
   total'index := 0;

   while total'index < max'supr'totals do
   begin
      total'offset := total'index * wl'supr'totals;
      @supr'totals':= @supr'totals(total'offset) & lsl(1);
      if supr'totals' <> blank'total,(bl'supr'totals) then
      begin
         if first'time then
         begin
            p "Totals:" output;
            first'time := false;
         end'if;
         print(supr'totals(total'offset),wl'supr'totals,0);
      end'if;
      total'index := total'index + 1;
   end'while;

end'proc;   <<print'totals>>
$page "call'suprtool2"
<<
  pass a command line to suprtool
>>

procedure call'suprtool2;
begin
  integer length;
```

```
  subroutine print'int(val);
    value val; integer val;
  begin
    val := ascii(val,10,outbuf');
    print(outbuf,-val,0);
    end'subr; <<print'int>>

  subroutine print'command;
  begin
    length := 256;
    do length:=length-1 until
    length<0 or b'command'line(length)<>" ";
    length:=length+1;
    print(supr'command'line,-length,0);
    end'subr; <<print'command>>

  suprtool2 (supr'control);
  if supr'status <> supr'ok then begin
    p "Suprtool2 interface call has failed!" err;
    print'command;
    p "Error number = " ask;
    print'int(supr'status);
    setjcw(-1);
    end'if;
  blank(supr'command'line,wl'supr'command'line);

  end'proc; <<call'suprtool2>>
$page "init'tool2spl"
<<
    check logon mode, parm values, etc.
    see if we can send one command to suprtool.
>>

logical procedure init'tool2spl;
begin
  integer temp;

  init'tool2spl := false;

  who(temp);
  batch := temp.(12:2) = 2;

  p "TOOL2SPL/Copyright Robelle 1982-2001"output;
  p current'version output;
  print(outbuf,0,0); <<blank line>>

  blank(supr'control,wl'supr'control);
  supr'version := 4;
  supr'maxdata := 0d;
  move supr'print'state := "ER";
  move supr'total'type  := "AS";
  if copy'of'parm=2 then
    move supr'print'state := "AL"
  else
  if copy'of'parm=3 then
    move supr'print'state := "NE";
  p "Please hit RETURN for EXIT at end. Thanks" output;
  move supr'command'line :=
    ":COMMENT Test of Suprtool Interface";
  call'suprtool2;
  if supr'status=0 then begin
    init'tool2spl := true;
    print(outbuf,0,0);
    end'if;

  end'proc;  <<init'tool2spl>>
$page "mainline"

  copy'of'parm := actual'parm;

  if init'tool2spl then begin
```

```
    << main loop of program >>
    do begin
      p ">" ask;
      input;
      if input'length<>0 then begin
        move b'command'line:=inbuf',(input'length);
        call'suprtool2;
        end'if;
      end'do
    until input'length=0 or supr'status<>0;
    move supr'command'line := "EXIT";
    call'suprtool2;
    if supr'status = 0 then
      print'totals;
    end'if;

end.  <<tool2spl>>
```

C Example

Below is a C program to invoke Suprtool through the interface routine.

```
#include <stdio.h>
#include <string.h>
/*
   DO NOT use int's in this structure, since they are the wrong size,
   and are aligned on 4-byte boundaries, which will confuse Suprtool2.
*/
struct supr_control
{
  short version;
  short status;
  char  command[256];
  char  priority[2];
  short maxdata1;
  short maxdata2;
  char  print_state[2];
  char  total_type[2];
  char  other_flags[18];
  short totals[15][9];
  short out_count1;
  short out_count2;
  char  workspace[20];
} SuprControl;

void suprtool2( struct supr_control *);
init_st2()
{
  int i;
  SuprControl.version        = 4;
  SuprControl.status         = 0;
  SuprControl.priority[0]    = 'C';
  SuprControl.priority[1]    = 'S';
  SuprControl.maxdata1       = 0;
  SuprControl.maxdata2       = 0;
  SuprControl.print_state[0] ='A';
  SuprControl.print_state[1] ='L';
  SuprControl.total_type[0]  = 'A';
  SuprControl.total_type[1]  = 'S';
  strcpy( SuprControl.workspace, "                    ");
  for (i=0;i<256;i++)
    SuprControl.command[i]=' ';
}


call_suprtool2()
{
  suprtool2( &SuprControl );
  if (SuprControl.status!=0)
  {
    puts("Suprtool2 interface call has failed!");
    printf("Error number = %d\n", SuprControl.status );
  }
}
assign_cmd( char *cmd )
{
  int i;
  for (i=0;i<256;i++)
    SuprControl.command[i]=' ';
  i=0;
  while (*cmd)
    SuprControl.command[i++]=*cmd++;
}
main()
{
  char cmd[256];
  init_st2();
  assign_cmd( "verify all");
  call_suprtool2();
  assign_cmd( "exit");
  call_suprtool2();
}
```

Pascal Example

Below is a Pascal program to invoke Suprtool through the interface routine. The example as given is for use with Pascal/iX. Note that the $HP3000_16$ compiler option is set. This is required because Suprtool2's control record uses 16-bit alignment instead of Pascal/iX's default 32-bit alignment.

For Pascal/V, change the $set nm$ line to $set 'nm=false'$

```
{ Example of Calling Suprtool2 from Pascal }
{ Set 'nm=false' for MPE V }
$set 'nm=true'$
$list off$
$if 'nm'$
  $HP3000_16$
$endif$

program test(input,output);

const
   MaxSuprCmdLen = 256;

type

$if 'not nm'$
   shortint = -32768 ..  32767;        (* 16-bit integer *)
$endif$

   SuprCmdStr   = string[MaxSuprCmdLen];
   SuprTotalStr = packed array[1..18] of char;

   SuprControlRec = record
 version    : shortint;
 status     : shortint;
      command    : packed array[1..MaxSuprCmdLen] of char;
      priority   : packed array[1..2] of char;
      maxdata    : integer;
      print_state: packed array[1..2] of char;
      total_type : packed array[1..2] of char;
      other_flags: packed array[1..18] of char;
      totals     : array[1..15] of SuprTotalStr;
      out_count  : integer;
      workspace  : packed array [1..20] of char;
   end;

var
   SuprControl : SuprControlRec;

procedure suprtool2( var control: SuprControlRec );
   external spl;

procedure InitST2;
var
   i: integer;
begin
   SuprControl.version        := 4;
   SuprControl.status         := 0;
   SuprControl.priority[1]    := 'C';
   SuprControl.priority[2]    := 'S';
   SuprControl.maxdata        := 0;
   SuprControl.print_state[1] := 'A';
   SuprControl.print_state[2] := 'L';
   SuprControl.total_type[1]  := 'A';
   SuprControl.total_type[2]  := 'S';
   for i:=1 to 20 do
      SuprControl.workspace[i] := ' ';
   for i:=1 to MaxSuprCmdLen do
      SuprControl.command[i] := ' ';
end;

procedure CallSuprtool2;
begin
  suprtool2( SuprControl );
  if (SuprControl.status<>0) then
  begin
    writeln('Suprtool2 interface call has failed!');
    writeln('Error number = ', SuprControl.status:1 );
  end;
end;
```

```
procedure ST2Cmd( cmd: SuprCmdStr );
var i: integer;
begin
  for i:=1 to MaxSuprCmdLen do
    SuprControl.command[i]:=' ';
  for i:=1 to strlen( cmd ) do
    SuprControl.command[i]:=cmd[i];
  CallSuprtool2;
end;

begin
  InitST2;
  ST2Cmd( 'in infile;out $null');
  ST2Cmd( 'exit');
  writeln( SuprControl.out_count );
end.
```

SPEEDWARE Example

You can call the Suprtool2 interface from a SPEEDWARE V.6 or V.7 program. Use the Supr-Control area to pass commands to Suprtool2.

There are 3 ways to get Speedware to find the Suprtool2 routine.

1. **Using Designer.** In the Operating Options (OPEO) screen add the XL to Local 4GL Parms. : XL="ST2XL.PUB.ROBELLE". This is the recommended approach for SPEEDWARE V.6.
2. **Using Linkedit.** Copy the routines into SPWXL.SPWxxxxx.SPEEDWRE. This is the XL that Speedware provides for this purpose. This could become a problem as SUPRTOOL2 changes over the years and the new version is not updated into SPWXL.
3. **Using RUN.** Add the XL to the run command. n.b. If you do this you will also need to include Speedware's database interface XL called DBMSXL.

```
:RUN SPW4GL.SPWxxxxx.SPEEDWRE
   ;XL="DBMSXL.SPWxxxxx.SPEEDWRE,ST2XL.PUB.ROBELLE"
```

This is the recommended approach for SPEEDWARE V.7 and higher.

Below is a sample SPEEDWARE process that calls the Suprtool2 interface. The program uses Suprtool to read a dataset, then writes the dataset to a file.

```
GLOBAL:
SET TITLE("Calling Suprtool Test");
EXIT;

MENU:
SET WINDOW(ROW=4,COL=4,HEIGHT=15,WIDTH=40);
BRANCH "Copy D-SALES to file"  DO REPORT-SUPRTOOL2;
EXIT;

REPORT-SUPRTOOL2:LISTING;
SET TITLE("Calling Suprtool2");
SET WINDOW;
DCL
  SUPR-CONTROL      FORMAT (TYPE = CHAR, BYTES =582),
  SUPR-VERSION      FORMAT (TYPE = INT , BYTES = 2, LEN= 4)
                    REDEFINES (SUPR-CONTROL[1:2]),
  SUPR-STATUS       FORMAT (TYPE = INT , BYTES = 2, LEN= 4)
                    REDEFINES (SUPR-CONTROL[3:4]),
  SUPR-COMMAND-LINE FORMAT (TYPE = CHAR, BYTES = 256)
                    REDEFINES (SUPR-CONTROL[5:260]),
  SUPR-PRIORITY     FORMAT (TYPE = CHAR, BYTES = 2)
                    REDEFINES (SUPR-CONTROL[261:262]),
  SUPR-MAXDATA      FORMAT (TYPE = INT , BYTES = 4, LEN= 9)
                    REDEFINES (SUPR-CONTROL[263:266]),
  SUPR-PRINT-STATE  FORMAT (TYPE = CHAR, BYTES = 2)
                    REDEFINES (SUPR-CONTROL[267:268]),
  SUPR-TOTAL-TYPE   FORMAT (TYPE = CHAR, BYTES = 2)
                    REDEFINES (SUPR-CONTROL[269:270]),
  SUPR-FUTURE-FLAGS FORMAT (TYPE = CHAR, BYTES = 18)
                    REDEFINES (SUPR-CONTROL[271:288]),
  SUPR-TOTALS  (15) FORMAT (TYPE = ZONE, BYTES = 18,)
                    REDEFINES (SUPR-CONTROL[289:558]),
  SUPR-OUT-COUNT    FORMAT (TYPE = INT , BYTES = 4, LEN= 9)
                    REDEFINES (SUPR-CONTROL[559:562]),
  SUPR-WORKSPACE    FORMAT (TYPE = CHAR, BYTES = 20)
                    REDEFINES (SUPR-CONTROL[563:582]);

(* initial values *)

CALCUL 4                      = SUPR-VERSION,
       0                      = SUPR-STATUS,
       'DS'                   = SUPR-PRIORITY,
       'ER'                   = SUPR-PRINT-STATE,
       'AS'                   = SUPR-TOTAL-TYPE,
       ' '                    = SUPR-WORKSPACE;

CALCUL 'base store,1,READER'  = SUPR-COMMAND-LINE;

CALL SUPRTOOL2            USING SUPR-CONTROL MODIFY;

IF SUPR-STATUS <> 0  THEN
   PRINT 'Error Suprtool', SUPR-STATUS, SUPR-COMMAND-LINE;
ELSE
BEGIN
   CALCUL 'get d-sales;out dsfile' = SUPR-COMMAND-LINE;

   CALL SUPRTOOL2         USING SUPR-CONTROL MODIFY;

END;

IF SUPR-STATUS <> 0  THEN
   PRINT 'Error Suprtool', SUPR-STATUS, SUPR-COMMAND-LINE;
ELSE
BEGIN
   CALCUL 'exit '                 = SUPR-COMMAND-LINE;

   CALL SUPRTOOL2         USING SUPR-CONTROL MODIFY;

END;
EXIT;
```

Calling Suprlink

The Suprtool2 Interface

User programs may "call" Suprlink via the Suprtool2 interface using the Suprtool Link command. A typical use of this interface would be for a COBOL program to ask Suprtool to extract selected subsets from two or more large IMAGE datasets and to use Suprlink to link the two files. The COBOL program would then read and format the Suprlink output file into a report.

Restrictions

When calling the Suprtool2 interface, you signal the end of commands and the start of processing by passing the Exit command. If you specify the Link command with no parameters, the interface switches to Suprlink. When you have passed all the Suprlink commands, do not end the Suprlink task by passing Exit. This will end the Suprlink task, but it will also result in a Suprtool error. There are two solutions to this problem:

1. Specify a substring of Exit (for example, E).
2. Specify all Suprlink commands with Suprtool's Link command (e.g., Link Exit).

COBOL Example

Below is a sample COBOL program named LINK2COB, which calls the Suprtool2 interface procedure. Its purpose is to print the notes for a selected product with the associated company name sorted by serial number. The program uses the Suprtool2 interface to do the following:

1. Select and sort records from the notes dataset.
2. Extract specific fields and sort the entire Customer dataset.
3. Use Suprlink to link the Notes and Customer files into the Notecust file.

The program then reads the Notecust file and prints a report on the line printer.

The database used in this example is called Cust, and the datasets are called Notes and Customer. Here is a Suprtool Form listing of the datasets:

```
:run suprtool.pub.robelle
>ba cust.history
>form notes
    NOTES            Detail
       Entry:
          SERIAL-NO            U4     (!CUSTOMER(SORT-ITEM))
          PRODUCT-CODE         U2
          TEXT-LINE            X80
          SORT-ITEM            K2
    Capacity: 75010  Entries: 40459  Entry Length: 45

>form customer
    CUSTOMER         Master
       Entry:
          SERIAL-NO            U4     <<Search-Field>>
          COMPANY-NAME         X60
          ALTERNATE-NAME       X60
          CONTACT-NAME         X40
          CONTACT-TITLE        X40
          PHONE-NO             X20
          CONTACT-NOTE         X40
          ADDRESS             4X40
          COUNTRY              X12
          CUSTOMER-NOTE        X40
          RELATED-NAME         X60
          SORTING-SEQ          U16
          OMNIDEX-SI           J2
          TIMESTAMP-INIT       U16
    Capacity: 1499  Entries: 1250  Entry Length: 286
```

The listing for the COBOL program is:

```
$control source,errors=15,nolist
$control debug,bounds
 identification division.
 program-id.    Link2cob.
 date-written.  15 Sep88.
 date-compiled.
 installation.  Robelle Solutions Technology Inc.
 security.      Copyright 1988-2001 Robelle Solutions Technology Inc.
 remarks.

*************************************************************
*                                                           *
*                     link2cob                              *
*                                                           *
*  functional overview:  Uses Suprtool to extract           *
*                        and print.                         *
*  input:  Notes dataset of the cust database.              *
*          All notes with a specified product               *
*          code and the related customer name               *
*          from the customer dataset.                       *
*                                                           *
*  output: print report on line-printer                     *
*                                                           *
*  terminated by:  eof or error                             *
*                                                           *
*  prep with:      cap = ph;maxdata=15000                   *
*                                                           *
*  rev#,init,date    , reason:                              *
*  01  , djg,15 Sep88, started first implementation.        *
*  02  , djg,10 Aug90, version 4 of the control record.     *
*                                                           *
*************************************************************

$page "environment division"
 environment division.
 configuration section.
 source-computer.  hp-3000.
 object-computer.  hp-3000.
 special-names.
     top is new-page.

 input-output section.
 file-control.
     select line-printer assign to "LINEPRT,,,LP".
     select notecust-file assign to "NOTECUST".

 data division.
 file section.
 fd  notecust-file
     data record is item-record.
 01  notecust-record.
     05  notecust-serial-no      pic x(4).
     05  notecust-product-code   pic x(2).
     05  notecust-text-line      pic x(80).
     05  notecust-company-name   pic x(60).

 fd  line-printer
     data record is line-record.
 01  line-record                 pic x(132).

$page "constants"
 working-storage section.
 01  true-value                  pic x value "T".
 01  false-value                 pic x value "F".
 01  revision-no                 pic 99             value 2.

$page "logical variables"
 01  end-of-notecust-flag        pic x.
     88  end-of-notecust         value "T".

$page "variables"
```

```
 01  if-command.
     05  filler                  pic x(17) value
                "IF PRODUCT-CODE='".
     05  if-product-code         pic x(2).
     05  filler                  pic x(2) value "'".

$page "input area"
 01  accept-buffer.
     05  input-buf               pic x(80).
         88  answer-spaces       value spaces.

$page "report layouts"
 01  report-header.
     05  filler                  pic x(9) value "Serial : ".
     05  head-serial-no          pic x(4).
     05  filler                  pic x(2) value spaces.
     05  head-company-name       pic x(60).

 01  report-detail.
     05  rept-text-line          pic x(80).

$page "suprtool control parameter"

$include cobol.qlibsrc.robelle

$page "procedure division"

************************************************************
*                                                          *
*          p r o c e d u r e    d i v i s i o n            *
*                                                          *
************************************************************

 procedure division.
 00-main                          section.

* ask for selection criteria.

     display "LINK2COB ", revision-no.
     display " ".

     display "Enter 2-character product code:".

     move spaces to input-buf.
     accept input-buf.
     if answer-spaces then go to 00-main-exit.

     move input-buf to if-product-code.

     perform 10-create-notecust-file
        thru 10-create-notecust-file-exit.

* Now, the file "notecust" contains the sorted data.
* This file must be read and printed on the line printer.

     perform 20-produce-report
        thru 20-produce-report-exit.

 00-main-exit.  goback.

$page "[10]  create-notecust-file"

 10-create-notecust-file       section.

     move "base cust.history,5" to supr-command-line.
                                perform 10-90-call-suprtool.

     perform 10-10-create-notefile.

     perform 10-20-create-custfile.
```

```
    perform 10-30-link-notefile-custfile.

    move "exit"                to supr-command-line.
                               perform 10-90-call-suprtool.

    go to 10-create-notecust-file-exit.

10-10-create-notefile.
    move "get notes"           to supr-command-line.
                               perform 10-90-call-suprtool.
    move if-command            to supr-command-line.
                               perform 10-90-call-suprtool.
    move "extract serial-no"   to supr-command-line.
                               perform 10-90-call-suprtool.
    move "extract product-code" to supr-command-line.
                               perform 10-90-call-suprtool.
    move "extract text-line"   to supr-command-line.
                               perform 10-90-call-suprtool.
    move ":purge notefile"     to supr-command-line.
                               perform 10-90-call-suprtool.
    move "output notefile,link"  to supr-command-line.
                               perform 10-90-call-suprtool.
    move "sort serial-no"      to supr-command-line.
                               perform 10-90-call-suprtool.
    move "sort sort-item,desc" to supr-command-line.
                               perform 10-90-call-suprtool.
    move "xeq"                 to supr-command-line.
                               perform 10-90-call-suprtool.

10-20-create-custfile.
    move "get customer"        to supr-command-line.
                               perform 10-90-call-suprtool.
    move ":purge custfile"     to supr-command-line.
                               perform 10-90-call-suprtool.
    move "output custfile,link"  to supr-command-line.
                               perform 10-90-call-suprtool.
    move "sort serial-no"      to supr-command-line.
                               perform 10-90-call-suprtool.
    move "extract serial-no"   to supr-command-line.
                               perform 10-90-call-suprtool.
    move "extract company-name" to supr-command-line.
                               perform 10-90-call-suprtool.
    move "xeq"                 to supr-command-line.
                               perform 10-90-call-suprtool.

10-30-link-notefile-custfile.
    move "link"                to supr-command-line.
                               perform 10-90-call-suprtool.
    move "input notefile by serial-no"
                               to supr-command-line.
                               perform 10-90-call-suprtool.
    move "link custfile"       to supr-command-line.
                               perform 10-90-call-suprtool.
    move ":purge notecust"     to supr-command-line.
                               perform 10-90-call-suprtool.
    move "output notecust"     to supr-command-line.
                               perform 10-90-call-suprtool.
    move "e"                   to supr-command-line.
                               perform 10-90-call-suprtool.

10-90-call-suprtool.
    call "SUPRTOOL2" using supr-control.
    if not supr-ok then
        display "Error:  Unable to call Suprtool2"
        display " "
        display "Suprtool interface error number: ",
                supr-status.

10-create-notecust-file-exit.  exit.

$page "[20]  produce-report"
```

```
*
* Read the notecust file which contains the detail records
* from the NOTES dataset and the company-name from the
* CUSTOMER dataset.  We produce a break on each new customer
* serial number; otherwise, we just print the text of the
* note lines.
*

 20-produce-report              section.

     open input  notecust-file.
     open output line-printer.

     move spaces to head-serial-no.
     move false-value to end-of-notecust-flag.

     perform 20-10-read-notecust
       until end-of-notecust.

     go to 20-produce-report-exit.

 20-10-read-notecust.
     read notecust-file at end
         move true-value to end-of-notecust-flag.
     if not end-of-notecust then
        perform 20-20-process-record.

 20-20-process-record.
     if notecust-serial-no <> head-serial-no then
        perform 20-30-print-header.
     move notecust-text-line to rept-text-line.
     write line-record from report-detail.

 20-30-print-header.
     move spaces to line-record.
     write line-record.
     move notecust-serial-no to head-serial-no.
     move notecust-company-name to head-company-name.
     write line-record from report-header.

 20-produce-report-exit.  exit.
```

Installing the Suprtool2 Interface

Compatibility Mode Installation

You do not need to do any installation steps in order to use the native mode version of Suprtool2. These instructions describe how to install the Suprtool2 interface into the System SL on Classic versions of MPE, and on MPE/iX for access by compatibility mode programs.

The job stream Suprcall.Suprjob.Robelle installs the Suprtool2 interface so that it can be used by any program on the system. Use it either to update the interface when you receive a new version of Suprtool or to re-install the interface after a MIT update from HP. You will need a small tape for a new COLDLOAD tape to contain the Suprtool interface segment. You can also install the Suprtool2 interface in a pub or group SL (see "Compiling and Linking on MPE V" on page 524).

Installing into the System SL

1. Make sure that the Robelle account has been created and all files have been restored.
2. Ensure that no one will use the interface until the installation is complete. No one can be running a program which uses the interface. Stop all jobs and send an operator warning.

```
:showjob
:warn @;please stop for 20 minutes
:abortjob #snnn
```

3. STREAM the installation job. Supply any passwords requested by the STREAM command.

```
:stream suprcall.suprjob.robelle
```

4. SUPR2JOB uses the SEGMENTER to add the Suprtool Interface Library into SL.Pub.Sys. It then requests a tape ("COLDLOAD") to create a new cold load tape containing MPE plus the Suprtool interface library. Mount a tape with a write ring and :REPLY. Save this tape and use it for any future cold loads.
5. If everything goes well, SUPR2JOB prints a final message on the console.
6. Please save the job listing for future reference.

Suprtool2 is now installed and you should be able to use it in your application programs.

Suprtool2 Error Messages

Error Numbers

Suprtool2 returns error numbers in the status parameter of the workspace. For most errors, a message is also displayed on $stdlist. The following summarizes the form of Suprtool2 error messages and the error numbers returned.

Messages On $Stdlist

Most Suprtool2 errors result in a message being displayed on $stdlist. All Suprtool2 error messages start with "ST2 -" making them easier to identify. For example,

```
Error:  ST2 - Caller lacks PH capability
```

1 - Unable to Access Files

Suprtool2 uses two temporary files called SI*nnn* and SL*nnn*. The *nnn* corresponds to the process identification number of the calling process. If Suprtool2 is unable to open, read, write, or close these files, a file system message is displayed on $stdlist and error #1 is returned.

2 - Suprtool Aborted

If the Suprtool task has any errors, Suprtool is terminated with the fatal JCW set. In this case, Suprtool2 returns error #2 to indicate that the requested task failed. If the print-state in the workspace was set to "ER" (error) or "AL" (always), the output from Suprtool will be displayed on $stdlist. If the print-state is "NE" (never), nothing is displayed on $stdlist.

3 - Unable to Create Suprtool Process

The Suprtool2 routine creates the Suprtool program as a son process. If this fails for any reason, error #3 is returned. The most common reason for this is forgetting to :PREP or :LINK your program with CAP=PH. If the Suprtool2filecommand JCW is set to 2, check that the :File command for Suprtool correctly specifies the name of the Suprtool program file.

4 - Invalid Total Type

The Suprtool2 control record has a field to indicate what kind of totals the interface should return. The valid values for this field are "CO" for COBOL totals and "AS" for Ascii totals. The total type must be in uppercase. This error is returned when the control record has been passed a value other than "CO" or "AS" for the total type.

5 - Unable to Create Suprtool Process

Suprtool2 calls the PROCINFO intrinsic to obtain the process identification number of the current process. If this routine fails for any reason, the error number returned from PROCINFO is printed on $stdlist and Suprtool2 returns error #5.

Glossary of Terms

Commonly-used Terms

Batch

Suprtool operates in **session** mode or **batch** mode. In batch, any error message causes Suprtool to quit. Warning messages do not cause an abort. If an error occurs, Suprtool sets the Job Control Word to flush the remainder of the job.

In batch mode, Suprtool does not prompt for missing information as it does in session mode. Instead, it attempts to choose the alternative that has the least chance of destroying valid data. For example, if the output file is a duplicate file name in batch mode, Suprtool saves the new output file with a "made up" name (Outputnn, where nn is from 00 to 20), prints a warning message, and aborts. Another example is that of missing database passwords. In session mode, Suprtool prompts for the password; in batch mode, it uses the CREATOR password instead.

Pseudo-Batch Tasks

During a canned on-line task, such as passing usefiles to Suprtool, you can "fool" Suprtool into responding YES to operational questions. For example, if one of the canned tasks requires Suprtool to `output myfile,erase`, then Suprtool asks the question

```
ERASE all records from this OUTPUT file [no]?
```

You can avoid typing "yes" in response to this question by invoking Suprtool with:

```
:run suprtool.pub.robelle; &
    stdin=$null; info="use filename"; parm=4
```

Blocksize

The block size of a file is the record length multiplied by the blocking factor. MPE permits block sizes up to 32,000 words, but Suprtool restricts the total block size. When copying an MPE file, the maximum block size of either the input or output file is 14,336 words. If Suprtool detects an input or output file with a block size larger than 14,336 words, it prints one of the following error message:

```
The input blocksize is greater than 14336 words
```

```
The output blocksize is greater than 14336 words
```

Calculator

Suprtool, Suprlink and Stexport and Dbedit treat any line that begins with an equal sign ("=") as an expression to be evaluated. To add two numbers together:

```
>=125+512

Result= 637.0
```

An expression consists of numbers and operators. The operators can be addition (+), subtraction (-), multiplication (*), division (/), or exponentiation (**). The value of the expression is printed immediately.

Any number can be followed by a percent sign (%). The calculator assumes that you want to qualify the number as a percentage. For example,

```
>=125*5%

Result= 6.25
```

A complete description of the Suprtool calculator is given after the description of the Xeq command.

Chains

A chain is a group of related records in an IMAGE dataset linked by a key value. The Chain command is used to force Suprtool to use IMAGE search-paths when reading the input source. The Chain command can also be used with master datasets, even though master datasets only have one record per key value.

Control Character

You create a control character by holding down the Control key while you strike another key. "Y" plus Control generates Control-Y. These are normally nonprinting characters, but they may do things to your terminal. For example, Control-G rings the bell.

Suprtool uses control characters for a number of purposes:

In the Before command, control characters specify the edit functions: Control-D for delete, Control-B for before, etc.

Control-Y stops execution of the current Suprtool task. Suprtool prints a status report and asks if you would like to stop the operation.

Control-H causes the cursor to backspace one position in the current line.

Control-X cancels the current input line.

Control-S pauses a listing that is printing too fast for you to read.

Control-Q resumes a listing that you have paused with Control-S.

Database

A database in Suprtool is an IMAGE/3000 database or an SQL database. A database is specified in the Base or Open command. Several commands (e.g., Get, Chain, or Select) do not work until a database has been specified. Some commands only work with IMAGE/3000 databases and other commands only work with SQL databases.

An IMAGE database consists of datasets (files) which in turn consist of fields. An SQL database consists of tables or views, and each table or view consists of columns. In Suprtool, a column name can be used anywhere that a field-name is used. The advantage of using a database is that information about the database is automatically available to Suprtool.

The Form command shows the database structure.

Dataset

A dataset is a file within an IMAGE database. The Form command provides path information in DBSCHEMA format, including entry length and blocking factor on the dataset description. Use Get dataset to access the data within a specific IMAGE dataset.

Datasets have an implied structure that you specify when you create the database. Use the Form command to obtain a list of fields in the current dataset. Fields are used by other Suprtool commands and they are described below.

Errors

Errors are messages printed by Suprtool indicating a fatal problem in the task which prevents it from completing. In batch mode, errors cause the job to terminate. Error messages are further described in Appendix A.

Field

A field is a portion of a record. When you access an IMAGE dataset, this makes Suprtool aware of the IMAGE fields in the dataset. When you access an SQL database with the Select command, Suprtool is aware of each column name (fields and columns are synonymous in Suprtool). The Define command allows you to define new fields or redefine existing fields to have new sizes or data-types. Use Define to get at bytes of interest within existing fields and to give them an appropriate name. Then you can refer to the defined field in other commands (e.g., Extract, If, etc.). The following commands all contain a field:

```
>if balance>10000
>sort account
>extract a,b
```

Filename

A **filename** is any valid filename and is used in Suprtool commands to identify the input source, specify the output destination, or to specify an external file to be accessed in the Table or Use command. File names may be enclosed in quotes. The following commands all contain file names:

```
>input xyz
>output *out
>use supruse
>input "xyz"
```

MPE/iX File Names

Suprtool accepts MPE/iX variables in file names. This allows users to create flexible file names using MPE/iX expressions. This features works in Suprtool/V or Suprtool/iX, but only when running on MPE/iX. Underscores are not permitted in variables that are to be substituted for file names. Any Suprtool command that accepts a file name supports variables in the file name. The following example creates an MPE/iX variable called 'today' with today's date in yymmdd format. Suprtool writes the output to a file that starts with the letter P and ends with today's date:

```
:setvar today hpyear * 10000 + hpmonth * 100 + hpdate
:run suprtool.pub.robelle
>input   mpefile
>output  P!today
>exit
```

MPE Command

If Suprtool does not recognize a command as one of its own, it treats the command as an MPE command. Examples of MPE commands are LISTF to get a list of your files; TELL to send messages to other users; SHOWTIME to get the date and time; SHOWJOB to get a list of other users on the computer, and STREAM to start a job.

```
>showjob
```

In Suprtool/V only MPE commands allowed in "break" mode are allowed. This means that commands such as RUN cannot be executed from within Suprtool/V. Suprtool/V will not execute any User Defined Commands or command files. Suprtool/iX can execute any MPE/iX command, UDC or command file.

POSIX File names

Suprtool accepts POSIX file names in any command that accepts MPE file names. File names starting with "/", "./", or "../" are treated as POSIX file names. All other file names are assumed to be MPE file names. MPE file names are upshifted and POSIX file names are not. POSIX file names are limited to a maximum of 239 characters. Here are some examples of POSIX file names:

```
:hello david,mgr.dev,david
:NEWDIR SUBDIR
:CHDIR SUBDIR
:run suprtool.pub.robelle
>input ./file
>verify input
/DEV/DAVID/SUBDIR/file
```

Strings

Suprtool expects all strings to be surrounded by a pair of single or double quotes (' or "). When Suprtool knows the length of a field, it pads strings with trailing spaces. For example,

```
>define  long,1,125       {125 character field}
>extract long="abcef"     {Suprtool adds 120 spaces}
>if      long="abcde"     {Suprtool checks for trailing spaces}
```

Suprtool accepts the null string. Suprtool pads it with spaces, so this is an easy way to see if a field is blank:

```
>if name = ""           {if name is blank}
```

One problem with any tool that accepts strings is how to include a quote mark inside the string. Suprtool offers two solutions:

1. Use the opposite quote mark (e.g., "don't").

2. Whenever two quote marks appear in a string, they are treated as a single quote (e.g., 'don"t').

Subscript

A subscript is used to specify one-of-many fields in a repeated item. Within IMAGE it is possible to specify repeated fields. For example:

```
costs,          5J2;
```

The item COSTS consists of five double integers. You select one element of a compound field by specifying a subscript in parentheses (the first element is 1, not 0). For example, if you wanted to select the input records where the second cost was greater than 10000, you would use:

```
>if costs(2) > 10000
```

The (2) portion of the command is the subscript. The default subscript is the first sub-item for Total, Define, Sort, and If, but the entire compound item for Extract. Table does not allow subscripts -- it always uses (1). The If command has another syntax, using up to three subscripts, allowing you to refer to subfields without Define (see the If command for details).

Tables

Tables are created with the Table command and they are used for testing in the If and Chain commands. Tables are used by the $lookup function of the If command. Use tables when you wish to check a data field for many different test values. You may also use tables to specify the records to search for with the Chain command.

Table can also mean a table from an SQL database.

Third-Party Indexing

In MPE/iX 4.0, Hewlett-Packard added support for third-party indexing (TPI) to TurboIMAGE/XL. TPI is a seamless interface to the indexing products of Dynamic Information Systems Corporation (DISC) and Bradmark Technologies Inc. Their products, Omnidex and Superdex, respectively offer generic and keyword search capabilities, through the standard interface in TurboIMAGE/XL.

TPI is only available to users on MPE/iX 4.0 or later. The database must be indexed using either Omnidex or Superdex, and must be enabled for indexing using DBUTIL.

If a database has TPI enabled, Suprtool can take advantage of the generic search capability in the Chain command. The Form command shows the third-party indexes that correspond to IMAGE fields and any byte-type indexes that do not correspond to an IMAGE field. MPE automatically maintains the indexes as Suprtool adds, deletes, and modifies records.

Warnings

Warnings are messages produced by Suprtool to let you know about nonfatal conditions that might affect your task. Some common warning messages and their meanings are described in Appendix A.

Yes or No

When Suprtool asks a question that requires a YES or NO answer, "Y", "OUI", "JA", and "SI" are accepted as "YES", and any other answer is considered "NO".

Special Characters

Special Characters

Certain non-alpha and non-numeric characters like > and : have special meaning within Suprtool. See the descriptions that follow. As well, the term "special" designates a class of characters in the If command.

* Means $Stdinx / $Stdlist or :File Command

* in the Input command means to read input from $stdinx (MPE only). * in the Output command means to write the output to $stdlist. * at the front of a file name points back to an MPE :File command.

```
>input *                                {MPE only}
>output *
>:file t;dev=tape                       {MPE only}
>input *t
```

= Means "Equals" or Calculate

```
= in the If command means "EQUALS":
>if customer = "40832"
= in commands means calculate something:
=10+25
Result= 35.0
```

= in the Input command means that the input file has exactly the same format and fields as the specified IMAGE dataset:

```
>input acctfile=actrans
= in the Output command means to write the sorted input file back into
itself.
>input myfile; key 1,10
>output=input                            {MPE only}
```

< Means "Less Than"

< in the If command means is one field "less than" another field or constant value:

```
>if balance < 10000
```

By combining < and =, you get "less than or equals":

```
>if balance <= 10000
```

> Means "Greater Than" or "Enter Command"

> is used for two purposes in Suprtool:

As the Suprtool prompt character (e.g., >Input actrec)

To mean *greater than* in an If command (e.g., if balance>10000). Combining > and =, gives >= for "greater than or equal to".

<> Means "Not Equals"

In the If command, use the two characters <> to mean "not equals":

```
>if status <> "01"
```

== Means "Matches Pattern"

In the If command, use the two characters == when you want to check a field for a pattern of characters. For example, to select records where the customer name contains the word "THOMPSON" somewhere, use:

```
>if name == "@THOMPSON@"
```

>< Means "Mismatches Pattern"

In the If command, use the two characters >< when you want to select records that fail to match a pattern of characters:

```
>if address >< "@CANADA@"
```

@ Means "Match Anything" in a Pattern

The At-Sign character (@) is used in patterns to indicate that Suprtool should accept anything in that position. For example:

```
>if name == "@ROBERT@"
```

The @ matches <null> ("ROBERT" is a valid match); it matches one character ("ROBERTA" is a valid match); it matches multiple characters ("ROBERT M. GREEN" and "The ROBERT E. LEE" are valid matches).

Means Number (in Patterns) or Prompt Character for the Edit Command

is used in patterns to match a single numeric character:

```
>if type=="REC##"  {look for "REC" followed by 2 digits}
```

is used by the Edit command on MPE to prompt for commands:

```
>edit                                   {MPE only}
#list d-inventory
```

is used in the Get and Input commands to read every *n*-th record

? Means Alphanumeric (in Patterns) or Prompt for Database Password

? is used in patterns to match a single alphabetic or numeric character:

```
>if type=="BASE??"{look for "BASE" plus 2 alphanumerics}
```

? is used in the Base and Put commands to force Suprtool to prompt for the database password in either session mode or batch mode:

```
>base store,5,?           {prompt for database password}
Password >                {password not echoed}
```

& Means Escape (in Patterns) or Continue Command Line

& is used in patterns to match one of the special pattern characters. For example, the #-character matches a single numeric character. If you need to look for the #-character itself, you would specify &# in the pattern:

```
>if type=="REC&#"       {look for "REC" followed by "#"}
```

& is used to continue a command line. You may enter commands on multiple input lines by putting an "&" continuation character at the end of the line:

```
>if status="20" and &          {continue the If command}
    state="AZ","CA","OR"       {select several states}
```

: Means O/S Commands or Bit Selection

Colon (:) at the start of a command line indicates an operating system command:

```
>:listf
>:ls
```

Colon (:) is used in the If command for bit selections:

```
>define bitfield,1,2,logical
>if bitfield.(4:2)=3
```

! Means O/S Commands

! at the start of a command line indicates an operating system command. This only works on Suprtool/UX.

```
>!du
>!ls
```

; Means Multiple Commands

Semicolon (;) is used to string several Suprtool commands together on a single line:

```
>input a;output b;xeq   {complete "task" is in one line}
```

, Means a List

Comma (,) in Suprtool commands is used to separate parameters:

```
>base actrec.data,3      {open the database exclusively}
>key 1,4,double          {specify a double-integer key}
>if acct=764523,456732,98765
```

Commas are optional in some Suprtool commands (e.g., Output), but are required in others (e.g., Extract).

, is the abbreviation for the Redo command.

„. is the abbreviation for the Do command.

„ is the abbreviation for the Listredo command.

" or ' Means String

Quotes (" or ') are the string delimiters in Suprtool (IF NAME="BOB"). Strings that start with " must end with ".

```
>if name='BOB'          {' is the string delimiter here}
```

(Means Start Parameter

Left parenthesis "(" is used to specify a subscript (see subscript below) or to select a specific range of input record numbers. Left parenthesis always comes with a right parenthesis.

```
>input actrec.data(10/20) {choose records 10 through 20}
>total budget(2)          {total second repeated field}
```

) Means End Parameter

Right parenthesis ")" is used to complete a subscript or a selected range of record numbers. Right parenthesis always comes with left parenthesis.

% Means Percentage

In the Numrecs command, use % to indicate the number of output records as a percentage of the input file size.

```
>numrecs 10%
```

/ Means Range of Records

Slash (/) in the Input and Get commands means a range of record numbers.

```
>input cat.dog.mouse(1000/2000)
```

\ Means Range of Fields

Backslash (\) in the Extract command means a range of fields.

```
>extract account \ rating
```

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z