



Revisions:

08/19/2009 – Corrected a deficiency in CLOSE.WORK.FILE not closing the DICT part of the file. This was preventing work files from being deleted by DELETE.WORK.FILE under certain circumstances.

08/05/2009 – Corrected a deficiency in checking the return status of TU.UPLOAD in PC.UPLOAD.

07/16/2008 – Added the REMEMBER and REMEMBER.OTHER.REC processes. Also added PC.DOWNLOAD.

06/28/2008 – Added the HOLD.DATA subroutine.

03/07/2008 – Added PC.UPLOAD subroutine.

12/02/2007 – Added F3.PC.FILE selection process.

10/12/2007 – Added SYSTEM49 subroutine/process.

09/14/2007 – Updated address in page footer.

09/01/2006 – Corrected MIN and MAX processes.

10/19/2004 – Added "autosense" delimiter option to IMPORT.DATA process; corrected P('..') syntax example on BUILD.KEY; added this "Revisions" section.

08/24/2004 – Added DYNAMIC.VALUE process.

08/22/2004 – Revised MIN and MAX processes.

08/20/2004 – Added UNIQUE process.

12/18/2003 – Created document

Precision Solutions, Inc.
Standard Implementation Processes
Last Updated: 19 Aug 2009





BUILD.KEY	Build a multiple part key into @KEY.
BUILD.KEY.VALUE	Build a multiple part key into @VALUE.

BUILD.KEY is a paragraph used in the PASA to build a multiple-part key from individual elements stored in a single @WORK attribute. The first part of the key is expected to be stored in @WORK<n,1>, the second in @WORK<n,2>, and so on.

BUILD.KEY.VALUE, by contrast, does the work for **BUILD.KEY** but it returns its result in @VALUE. This is useful when you need to build a multiple part key, but not assign it as the actual key to the current record.

From a paragraph:

```
EXEC 'BUILD.KEY,n,m,d'
L.KEY = P('BUILD.KEY,n,m,d')
```

n = The @WORK attribute where the key parts are stored.
 m = The number of elements to use for this key.
 d = delimiter to use between elements.

BUILD.KEY

```
@KEY ,, P BUILD.KEY.VALUE )
```

BUILD.KEY.VALUE

```
LOCAL L.AMC, L.PART.CNT, L.DELIM
*
L.AMC      ,, FIELD @PARAM, ' , ' , )
L.PART.CNT ,, FIELD @PARAM, ' , ' , 2)
L.DELIM    ,, FIELD @PARAM, ' , ' , ?)
*
IF L.DELIM ,, ' ' ) THEN
    L.DELIM ,, ' *'
END
*
@KEY ,, ' '
WHILE L.PART.CNT DO
    @KEY ,, @WORK L.AMC, L.PART.CNT > : @KEY
    *
    IF L.PART.CNT > ) THEN
        @KEY ,, L.DELIM : @KEY
    END
    *
    L.PART.CNT ,, L.PART.CNT -
REPEAT
```



CREATE.WORK.FILE	Create a temporary (work) file
<p>CREATE.WORK.FILE is a paragraph that creates a new file called <i>WRK.port</i> that can be used as a temporary holding place for information. This is most commonly used when generating information to be shown on a report.</p> <p>The optional single parameter defines the size of the file to be created. If omitted, a modulo of 101 is used. If the file exists when this process is called, the file is not resized, but the contents will be cleared.</p> <p>Dependents: CLOSE.WORK.FILE, CLEAR.WORK.FILE</p>	

```

LOCAL L.SIZE,L.CMD,L.NDX
*
L.SIZE ,, @PARAM
IF NOT NUM(L.SIZE) OR (L.SIZE < 0 ) THEN
  L.SIZE ,, 0
END
*
* Create the work file
*
L.CMD ,, '>:CREATE.FILE WRK.' : @PORT : ' ' : L.SIZE
EXEC L.CMD
*
* Remove the work file from the list of opened files
*
EXEC ' CLOSE.WORK.FILE'
*
* In the event the CREATE.FILE was unsuccessful (because the work file
* was already there), let's clear out the contents so we can be assured
* of starting with a clear file.
*
EXEC ' CLEAR.WORK.FILE' ;* In case the file was already there
  
```



CLEAR.WORK.FILE	Clear the contents of the work file
<p>CLEAR.WORK.FILE is a paragraph that clears the file named WRK.<i>port</i>. It is called as a part of CREATE.WORK.FILE. It can also be called independently when you have multiple uses of the work file between the time the file is created and the time the file is deleted.</p>	
<p>This process uses no parameters.</p>	

```
EXEC ' >:CLEAR.FILE DATA WRK.' :@PORT
```

CLOSE.WORK.FILE	Close a work file
<p>SB+ keeps a list of open file handles in the @FILES.OPENED variable. When using work files, where the file may be created and deleted several times throughout the course of a logon session, this variable must be updated to tell SB+ that the file is no longer available. Otherwise, SB+ will keep an open file handle to a file that may have been deleted.</p>	
<p>This paragraph closes the file named WRK.<i>port</i> (as created via CREATE.WORK.FILE) and uses no parameters.</p>	

```
LOCAL L.NDX
*
* Remove the work file from the list of opened files
*
CLOSE ' WRK.' :@PORT
CLOSE ' DICT WRK.' :@PORT
*
* For earlier versions of SB+ where CLOSE may not work
*
L.NDX ,, LOC ↓ WRK.' :@PORT, @FILES.OPENED, @AM)
IF L.NDX THEN
  @FILES.OPENED L.NDX> ,, ''
END
*
L.NDX ,, LOC ↓ DICT WRK.' :@PORT, @FILES.OPENED, @AM)
IF L.NDX THEN
  @FILES.OPENED L.NDX> ,, ''
END
```



DELETE.WORK.FILE	Delete the work file.
<p>DELETE.WORK.FILE is a paragraph that deletes the work file from the system. Note that this paragraph uses the Unidata "FORCE" extension that may be incompatible with other operating environments. For other operating environments, omit this detail.</p> <p>This process uses no parameters.</p>	

```
EXEC ' CLOSE.WORK.FILE'  
EXEC ' >:DELETE.FILE WRK.' :@PORT:' FORCE'
```



DYNAMIC.VALUE	Calculate the value of an expression stored in a string.
<p>DYNAMIC.VALUE takes an expression string, evaluates the expression, and returns the result. This allows you to incorporate SB+ expressions into your programming without having to directly interface to the SB+ subroutines for expression evaluation.</p> <p>Example:</p> <pre>LOCAL L.EXPRN, L.RESULT L.EXPRN ,, "@WORK 5 > : @WORK 2 >" L.RESULT ,, P_"DYNAMIC.VALUE," : L.EXPRN)</pre> <p>This subroutine can be called with either one or two (comma-delimited) parameters following the process name. If one parameter is provided, that parameter is expected to be the expression to be evaluated. If two parameters are used, the first parameter names the file where field names used in the expression can be found, and the second parameter is the expression itself. The result of the expression is returned in the common variable @VALUE.</p> <p>Note: For static expressions that are to be evaluated in a batch process like a report, update, or export, this kind of tool is not the most efficient approach to the problem. Rather, it would be better in that situation to call SB.STACK.EXP once for each expression to be used, and then call SB.EVAL.EXP alone with the expression stacks for each field when processing each input record.</p> <p style="text-align: right;">08/24/2004</p>	

Precision Solutions, Inc.
 Standard Implementation Processes
 Last Updated: 19 Aug 2009



```

SUBROUTINE DYNAMIC.VALUE
*
* Written By: Kevin King
* Date: 27 Aug 2004
* Project: Standard
* Description: This subroutine will accept a string in the common variable
* @PARAM and will interpret and return the result of that
* string, if it's syntactically valid.
*
*****
* Modifications
*****
* Date..... Changed By..... Description of Change.....
*
*****
* Include_s)
*****
$INCLUDE DMSKELCODE COMMON
*
DICT.NAME ,, FIELD_PARAM,' ', )
EXPRN ,, FIELD_PARAM,' ',2)
ERR ,, 0
*
IF _DICT.NAME NE '' ) AND _EXPRN EQ '' ) THEN
  F.DICT ,, F.MD
  EXPRN ,, DICT.NAME
END ELSE
  CALL SB.OPEN.FILE_"DICT " : DICT.NAME,F.DICT,ERR)
END
*
VALUE ,, '' ;* Initialize the return value
*
IF _ERR) THEN
  RTN.FLAG ,,
END ELSE
  STACK ,, ''
  ERR ,, 0
  CALL SB.STACK.EXP_EXPRN, F.DICT, STACK, ERR)
  *
  IF _ERR) THEN
    RTN.FLAG ,,
  END ELSE
    CALL SB.EVAL.EXP_STACK)
  END
END
*
RETURN
  
```



EF3	Edit the intuitive help process.
<p>EF3 is a very useful debugging paragraph. When using a screen for data entry, move the cursor to a prompt that has a process called for intuitive help, enter EF3 and that process will appear in its appropriate editor. You can then change the process, escape back to the screen, and continue testing all without ever having to go back into the screen definition tool.</p> <p>This process uses no parameters.</p>	

```
LOCAL L.SEL.PROC
*
L.SEL.PROC ,, @LINE , 7>"G [ "
IF L.SEL.PROC # '' ) THEN
  L.SEL.PROC ,, L.SEL.PROC"G, " ;* Cut off the parameter
  @RTN.FLAG ,, 0
  READ @OTHER.REC FROM @SYSID : ' PROCESS' ,L.SEL.PROC
  IF @RTN.FLAG ,, 0 THEN
    DATA L.SEL.PROC,"@?5"
    EXEC 'MP' ;* Modify process
  END
END
END
```



EFD	Edit the field definition
<p>EFD is a very useful debugging paragraph. When using a screen for data entry, move the cursor to a prompt and enter EFD and the field definition for the current field will appear. You can then view or amend the values shown there. Note that if you change anything here, you will need to escape the running screen (upon return from the field definition tool) and restart the screen process to have SB+ regenerate the drivers for that screen.</p> <p>This process uses no parameters.</p>	

```

EQU FIELD.NAME TO @LINE , 8>
IF @MAINFILE # '' AND FIELD.NAME # '' THEN
    DATA @MAINFILE, FIELD.NAME
    EXEC 'FD' ;* Field Defn
END
    
```

EV	Edit the validation process.
<p>EV is a very useful debugging paragraph. When running a screen, move the cursor to a prompt that uses the C: validation to call a process. Enter /EV on that prompt and the validation process will appear in the appropriate editor. You can then change this process, escape back to the running screen, and test your changes all without ever going back into the screen definitions tool.</p> <p>This process uses no parameters.</p>	

```

LOCAL VAL.PROC
VAL.PROC .. @LINE , ?>"G{ "
IF VAL.PROC[ ,2] .. 'C:' ) THEN
    VAL.PROC .. VAL.PROC[?,?2200]
    DATA VAL.PROC, "@?5"
    EXEC 'MP'
END
    
```



F3.PC.FILE	Select PC File via Windows File Open
<p>This BASIC subroutine interfaces with the standard TU.FORM.OPENDOS subroutine and allows the selection of a PC file via the standard Windows File Open dialog.</p> <p>To use this, create the subroutine and a matching /PD.B, then when you want to use F3 to select a file on the workstation/PC, use this:</p> <pre>F?.PC.FILE, path, desc, ext</pre> <p>As to the variables, <i>path</i> is the path you want to start in (like C:\Temp) <i>desc</i> is the description of the files you want to select (like Comma Separated Value) and <i>ext</i> is the extension you want to select (like CSV – note: no dot). So the whole thing all together might look like this:</p> <pre>F?.PC.FILE,C:\TEMP,Comma Separated Value,CSV</pre> <p>The fully qualified file name (with the drive letter, colon, and path) is returned.</p> <p style="text-align: right;">12/02/2007</p>	

```

SUBROUTINE F?.PC.FILE
*
$INCLUDE DMSKELCODE COMMON
*
START.PATH ,, FIELD_PARAM,' ', )
FILE.DESC ,, FIELD_PARAM,' ',2)
FILE.EXT ,, FIELD_PARAM,' ',?)
RESULT ,, ''
FILE.ERROR ,, ''
CALL TU.FORM.OPENDOS "", START.PATH, FILE.DESC:@SM:"*":FILE.EXT,
RESULT, FILE.ERROR)
*
IF NOT FILE.ERROR) THEN
  VALUE ,, RESULT
END ELSE
  VALUE ,, ''
END
*
RETURN
  
```



GN	Generate new process name
<p>Every process must have a name, and naming each process uniquely can be a time-consuming and mentally challenging process. The GN paragraph removes both of these limitations by having SB+ assign the next available process name based on a prefix and sequentially assigned process number for each prefix.</p> <p>To use GN, move the cursor to the Process Name prompt of any tool and enter <code>/GN,prefix</code>. The prefix can be anything, but as you can see from the source code, if a prefix is not provided, my process uses the default prefix of "PSI" (Precision Solutions, Inc.). You may want to change this to match your own preferred default prefix. Also, if the prefix is prefaced by a "?" character, the last process generated for that prefix will be recalled, instead of assigning a new process name.</p>	

```
LOCAL PREFIX
*
PREFIX ,, @PARAM
IF PREFIX[ , ] ,, '?' THEN
  PREFIX ,, PREFIX[2,999]
  IF PREFIX ,, '' THEN PREFIX ,, 'PSI'
  READ @OTHER.REC FROM 'PRPREFIX',PREFIX
END ELSE
  IF PREFIX ,, '' THEN PREFIX ,, 'PSI'
  READ @OTHER.REC FROM 'PRPREFIX',PREFIX
  O ,, O +
  WRITE @OTHER.REC ON 'PRPREFIX',PREFIX
END
*
DATA PREFIX : ↻ "MR%4")
```



HOLD.DATA	Capture DATA stacked values, call a process, and restack the values
<p>This passthrough subroutine captures DATA stacked values, calls a process, and then restacks the data that was captured. This is useful when integrating into a vendor application that is doing extensive DATA stacking that would otherwise impede the modification.</p> <p style="text-align: right;">06/27/2008</p>	

```

SUBROUTINE HOLD.DATA
*
* Written By: Kevin King
* Project: Standard
* Date: 27 Jun 2008
* Description: This subroutine captures information that has been data
* stacked, calls a process, and then restacks the data.
*
*****
* Modifications
*****
* Date..... Changed By..... Description of Change.....
*
*****
* Include(s)
*****
$INCLUDE DMSKELCODE COMMON
*
PROCESS.NAME ,, PARAM
*
DATA.STACK ,, ''
DATA.CNT ,, ''
*
LOOP
WHILE (SYSTEM_0)) DO
  INPUT DATA.VAL
  DATA.CNT +,
  DATA.STACK ,DATA.CNT> ,, DATA.VAL
REPEAT
*
CALL SB.PROCESS_PROCESS.NAME)
*
FOR DATA.LOOP ,, TO DATA.CNT
  DATA DATA.STACK ,DATA.LOOP>
NEXT DATA.LOOP
*
RETURN
  
```



IF.AMEND	Call a process from a screen, but only if amending a record.
This passthrough paragraph is typically called from a screen definition either from a function key, from the PASA, or from the PAU. It checks to see if the record is being amended, and if so, calls the process listed in the parameter.	

```
IF @ACTION = 2 THEN  
  EXEC @PARAM  
END
```



IF.DEL	Call a process from a screen, but only if deleting a record.
This passthrough paragraph is typically called from a screen definition either from the PASA, or from the PAU. It checks to see if the record is being deleted, and if so, calls the process listed in the parameter.	

```
IF @ACTION ,, ? THEN  
    EXEC @PARAM  
END
```

IF.NEW	Call a process from a screen, but only if inserting a new record.
This passthrough paragraph is typically called from a screen definition either from a function key, from the PASA, or from the PAU. It checks to see if the record is new, and if so, calls the process listed in the parameter.	

```
IF @ACTION ,, THEN  
    EXEC @PARAM  
END
```



IMPORT.DATA	Import data from a flat (sequential) file.
<p>This subroutine is typically called from another process that sets up the parameters. It opens a sequential file and for each record loaded, it formats the record into multiple attributes and then calls a process that can be used to consume the loaded record.</p>	
<p>This process accepts four comma-separated parameters from the common variable @PARAM:</p> <ul style="list-style-type: none"> • The directory file (DIR-type file pointer in the current account) where the sequential item can be found; • The name of the sequential item in the directory file; • The delimiter used between fields on each row of the import data: C=Comma, T=Tab, ?=Autosense, all other characters as-is; • The process to be called for each loaded record. <p style="text-align: right;">Revised 10/19/2004</p>	

```

SUBROUTINE IMPORT.DATA
*
* Written By: Kevin King
* Date:      20 Aug 2004
* Project:   Standard
* Description: This subroutine will import information from a sequential
*             file and will call a process for each imported row.
*
*****
* Modifications
*****
* Date..... Changed By..... Description of Change.....
*
$INCLUDE DMSKELCODE COMMON
*
TRUE  ,, EQ  )
FALSE ,, NOT TRUE)
*
IMPORT.DIR  ,, FIELD_PARAM,' ', )
IMPORT.NAME ,, FIELD_PARAM,' ',2)
IMPORT.DELIM ,, FIELD_PARAM,' ',?)
IMPORT.PROC ,, FIELD_PARAM,' ',4)
*
IF _IMPORT.DIR NE '' ) THEN
  IF _IMPORT.NAME NE '' ) THEN
    IF _IMPORT.PROC NE '' ) THEN
      BEGIN CASE
        CASE _IMPORT.DELIM EQ ' C ' )
          IMPORT.DELIM ,, ' ,'
        CASE _IMPORT.DELIM EQ ' T ' )
          IMPORT.DELIM ,, CHAR_9)
        CASE _IMPORT.DELIM EQ ' ' ' )
          IMPORT.DELIM ,, ' ,'
      END CASE
    END IF
  END IF
END IF

```

Precision Solutions, Inc.
Standard Implementation Processes
Last Updated: 19 Aug 2009



```

END CASE
*
OPENSEQ IMPORT.DIR,IMPORT.NAME TO F.IMPORT THEN
  GOSUB 000 ;* Process the import file
  CLOSESEQ F.IMPORT
END ELSE
  CALL SB.DISP_?,IMPORT.NAME : ' [E ]' )
END
END ELSE
  CALL SB.DISP_?,' Import process name is unexpectedly missing.' )
END
END ELSE
  CALL SB.DISP_?,' Import file name is unexpectedly missing.' )
END
END ELSE
  CALL SB.DISP_?,' Import directory file name is unexpectedly missing.' )
END
*
RETURN
*
*****
000 * Process the import file
*****
*
CALL SB.DISP_8,' Processing input file' )
*
EOF .. FALSE
RTN.FLAG .. 0
*
LOOP
  READSEQ BLOCK FROM F.IMPORT ELSE EOF .. TRUE
  UNTIL EOF OR _RTN.FLAG EQ ' X' ) DO
    CONVERT CHAR_0) : CHAR_?) TO '' IN BLOCK
  *
  IF _BLOCK NE '' ) THEN
    IF _IMPORT.DELIM EQ '?' ) THEN
      IMPORT.DELIM .. TRIM_0CONV_0CONV_BLOCK,' MC/A' ),' MC/N' ))[ , ]
    END
  *
  RECORD .. CHANGE_BLOCK, IMPORT.DELIM,@AM)
  CALL SB.PROCESS_IMPORT.PROC)
END
REPEAT
*
RETURN

```



MIN	Calculate the smallest value in a multivalued list.
<p>This paragraph takes a multivalued list of values in @PARAM and returns the smallest value in the list. Note that because it uses @PARAM as an input value, the input cannot exceed 999 characters (which is the maximum number of characters passed by SB+ in @PARAM). If this limitation is troublesome, you can create a version of this that accepts input from the common variable @VALUE which has no such limitation.</p> <p style="text-align: right;">Revised 09/01/2006</p>	

```

LOCAL L.VALUE, L.NDX, L.THIS
*
L.VALUE ,, @PARAM
L.NDX ,, DCOUNT L.VALUE, @VM)
@VALUE ,, L.VALUE , >
*
WHILE L.NDX > ) DO
  L.THIS ,, L.VALUE , L.NDX>
  IF L.THIS # '' ) THEN
    IF L.THIS @VALUE) THEN
      @VALUE ,, L.THIS
    END
  END
END
*
L.NDX ,, L.NDX -
REPEAT
  
```



MAX	Calculate the biggest value in a multivalued list.
<p>This paragraph takes a multivalued list of values in @PARAM and returns the biggest value in the list. Note that because it uses @PARAM as an input value, the input cannot exceed 999 characters (which is the maximum number of characters passed by SB+ in @PARAM). If this limitation is troublesome, you can create a version of this that accepts input from the common variable @VALUE which has no such limitation.</p> <p style="text-align: right;">Revised 09/01/2006</p>	

```

LOCAL L.VALUE, L.NDX, L.THIS
*
L.VALUE ,, @PARAM
L.NDX ,, DCOUNT L.VALUE, @VM)
@VALUE ,, L.VALUE , >
*
WHILE L.NDX > ) DO
  L.THIS ,, L.VALUE , L.NDX>
  IF L.THIS # '' ) THEN
    IF L.THIS > @VALUE) THEN
      @VALUE ,, L.THIS
    END
  END
END
*
L.NDX ,, L.NDX -
REPEAT
  
```



PC.DOWNLOAD	Transfer a file from the host to the workstation.
<p>This BASIC subroutine wraps around the TU.DOWNLOAD routine and provides a generic mechanism for transferring a flat or delimited file from the host to the workstation. It is executed from a paragraph via:</p> <p>EXEC "PC.DOWNLOAD,<i>hostFile</i>,<i>hostId</i>,<i>pcFile</i>"</p> <p>...where <i>hostFile</i> is the name of a file on the host where the information is stored, <i>hostId</i> is the key to the record on the host, and <i>pcFile</i> is the fully qualified name of the file to be stored on the workstation. This syntax also assumes that the code has been installed and a /PD.B wrapper (with the same name) has been created. If no /PD.B wrapper is used, substitute "B:PC.DOWNLOAD" for "PC.DOWNLOAD" in the above syntax.</p> <p style="text-align: right;">07/16/2008</p>	

```

SUBROUTINE PC.DOWNLOAD
*
* Written By: Kevin King
* Date: 20 Apr 2000
* Project: 0005?
* Description: This subroutine will transfer an item from Unix to the local
* PC (usually to be edited by Excel).
*
*****
* Modifications
*****
* Date..... Changed By..... Description of Change.....
* 25 Oct 0 Kevin King Added more descriptive error messages.
*
*****
* Include(s)
*****
$INCLUDE DMSKELCODE COMMON
*
FILENAME ,, FIELD_PARAM,' ',' )
SEQNAME ,, FIELD_PARAM,' ',' 2)
PCNAME ,, FIELD_PARAM,' ',' ?)
*
OPTS ,, ' H'
DESC ,, ' Transferring Data'
STATUS ,, ''
*
OPEN FILENAME TO F.QF THEN
  READ ITEM.QF FROM F.QF,SEQNAME THEN
    CALL TU.DOWNLOAD_ITEM.QF,PCNAME,OPTS,DESC,STATUS)
  IF STATUS NE 0) THEN

```

Precision Solutions, Inc.
Standard Implementation Processes
Last Updated: 19 Aug 2009



```
GOSUB 00
RTN.FLAG ,,
END
END ELSE
CALL SB.DISP_?, 'Unable to read ' : SEQNAME : ' for transfer' )
END
END ELSE
CALL SB.DISP_?, 'Unable to open ' : FILENAME : ' for transfer' )
RTN.FLAG ,,
END
*
RETURN
*
00 * Output an error message
*
BEGIN CASE
CASE _STATUS EQ 0)
MSG ,, '' ; * No error
CASE _STATUS EQ )
MSG ,, 'The transfer has been cancelled by the user.'
CASE _STATUS EQ 2)
MSG ,, 'The PC file cannot be opened. It may be in use or marked read-only.'
CASE _STATUS EQ ?)
MSG ,, 'There was an error reading the PC file.'
CASE _STATUS EQ 4)
MSG ,, 'Unknown communications error.'
CASE _STATUS EQ 5)
MSG ,, 'Retry limit exceeded.'
CASE _STATUS EQ 6)
MSG ,, 'Error writing to the PC.'
CASE _STATUS EQ 7)
MSG ,, 'File transfer not supported.'
CASE _STATUS EQ )
MSG ,, 'File exists, and overwrite has not been specified'
END CASE
*
IF _MSG NE '' ) THEN
MSG ,, 'Transfer failed: ' : MSG
CALL SB.DISP_?, MSG)
END
*
RETURN
```



PC.UPLOAD	Transfer a file from the workstation to the host.
<p>This BASIC subroutine wraps around the TU.UPLOAD routine and provides a generic mechanism for transferring a flat or delimited file from the workstation to the host. It is executed from a paragraph via:</p> <p>EXEC "PC.UPLOAD,<i>pcFile</i>,<i>hostFile</i>,<i>hostId</i>"</p> <p>...where <i>pcFile</i> is the fully qualified name of the file on the workstation, <i>hostFile</i> is the name of a file on the host where the information will be stored, and <i>hostId</i> is the key to the record. This syntax also assumes that the code has been installed and a /PD.B wrapper (with the same name) has been created. If no /PD.B wrapper is used, substitute "B:PC.UPLOAD" for "PC.UPLOAD" in the above syntax.</p> <p style="text-align: right;">03/07/2008</p>	

```

SUBROUTINE PC.UPLOAD
*
* Written By: Kevin King
* Date: 04 Dec 2007
* Project: Standard
* Description: This subroutine will wrap around TU.UPLOAD and will transfer
* a document from the workstation to a local file.
*
*****
* Modifications
*****
* Date..... Changed By..... Description of Change.....
*
*****
* Include_s)
*****
$INCLUDE DMSKELCODE COMMON
*
*****
* Parse Parameters
*****
PC.NAME ,, FIELD_PARAM,' ', )
OUT.FILE ,, FIELD_PARAM,' ', 2)
OUT.NAME ,, FIELD_PARAM,' ', ?)
*
*****
* Open the output file
*****
OPEN OUT.FILE TO F.OUT THEN
    GOSUB 000 ;* Load the data
    
```

Precision Solutions, Inc.
Standard Implementation Processes
Last Updated: 19 Aug 2009



```
END ELSE
  CALL SB.DISP_?,OUT.FILE : ' [E ]' )
  RTN.FLAG ,,
END
*
RETURN
*
*****
000 * Load the data
*****
*
OPTS  ,, ' H'
DESC  ,, ' Transferring File'
STATUS ,, 0
*
CALL TU.UPLOAD_PC.NAME,F.OUT,OUT.NAME,OPTS,DESC,STATUS)
IF _STATUS NE 0) THEN
  RTN.FLAG ,,
END
*
RETURN
```



REMEMBER	Store a copy of the current record before updating it (just in case!)
<p>This paragraph will store a copy of the current record in the REMEMBERED file (a file created specifically for this purpose) so you can easily restore information in the event that an update (via IMPORT.DATA or a periodic update) needs to be backed out.</p> <p>The REMEMBER process takes three parameters:</p> <ul style="list-style-type: none"> • A "set" name (where a "set" is a group of records that are being updated together. If not provided the current date/time is used for the set name in the form yyymmddhhmmss (year, month, day, hours, minutes, seconds). • The name of the file where this information came from. If not provided, @MAINFILE is used instead. • The key to the record that is being updated. If not provided, @KEY is used instead. <p>To use the default parameters, it is important to be sure to pass a null parameter with the trailing comma, as in: EXEC 'REMEMBER,'</p> <p>These three parameters are then used to build the key to the REMEMBERED file.</p>	
07/16/2008	

```

LOCAL L.SET.NAME,L.FILE.NAME,L.KEY,L.REMEMBERED.ID
*
L.SET.NAME ,, FIELD @PARAM,' ',' )
L.FILE.NAME ,, FIELD @PARAM,' ',' ,2)
L.KEY      ,, @PARAM"G2,9"
*
IF L.SET.NAME ,, '' THEN
    L.SET.NAME ,, @DATE"D4Y" : @DATE"DM" : @DATE"DD" : @TIME"MTS")"MCN"
END
*
IF L.FILE.NAME ,, '' THEN
    L.FILE.NAME ,, @MAINFILE
END
*
IF L.KEY ,, '' THEN
    L.KEY ,, @KEY
END
*
L.REMEMBERED.ID ,, L.SET.NAME : ' %' : L.FILE.NAME : ' %' : L.KEY
WRITE @RECORD ON 'REMEMBERED',L.REMEMBERED.ID
    
```



REMEMBER.OTHER.REC	Store a copy of @OTHER.REC before updating it (just in case!)
<p>This SB+ paragraph is a copy of REMEMBER except that it writes @OTHER.REC to the REMEMBERED file.</p> <p>This process, like REMEMBER, takes three parameters:</p> <ul style="list-style-type: none"> • A "set" name (where a "set" is a group of records that are being updated together. If not provided the current date/time is used for the set name in the form yyymmddhhmmss (year, month, day, hours, minutes, seconds). • The name of the file where this information came from. If not provided, @MAINFILE is used instead. • The key to the record that is being updated. If not provided, @KEY is used instead. <p>To use the default parameters, it is important to be sure to pass a null parameter with the trailing comma, as in: EXEC 'REMEMBER.OTHER.REC,'</p> <p>These three parameters are then used to build the key to the REMEMBERED file.</p> <p style="text-align: right;">07/16/2008</p>	

```

LOCAL L.SET.NAME,L.FILE.NAME,L.KEY,L.REMEMBERED.ID
*
L.SET.NAME  ,, FIELD @PARAM,' ',' )
L.FILE.NAME  ,, FIELD @PARAM,' ',' ,2)
L.KEY       ,, @PARAM"G2,9"
*
IF L.SET.NAME  ,, '' THEN
    L.SET.NAME  ,, @DATE"D4Y" : @DATE"DM" : @DATE"DD" : @TIME"MTS")"MCN"
END
*
IF L.FILE.NAME  ,, '' THEN
    L.FILE.NAME  ,, @MAINFILE
END
*
IF L.KEY  ,, '' THEN
    L.KEY  ,, @KEY
END
*
L.REMEMBERED.ID  ,, L.SET.NAME : ' %' : L.FILE.NAME : ' %' : L.KEY
WRITE @OTHER.REC ON 'REMEMBERED' ,L.REMEMBERED.ID
    
```



SELECT.TO.LIST	Execute OE SELECT/SSELECT statements and save a list.
<p>This paragraph adds a number of OE-based selection abilities to SB+, including:</p> <ul style="list-style-type: none"> • Cascading selects • SAVING UNIQUE support • Sorting and selection not otherwise supported by /PD.S <p>@VALUE is sent in as the parameter with a mv'd list of SELECT/SSELECT statements to be executed. The end result is either a saved list named WRK.<i>port</i> or the absence of the list (if nothing was selected).</p>	

```

LOCAL L.PROC, NDX, MAX
*
L.PROC      ,, ' PQ
L.PROC - > ,, ' HDELETE.LIST WRK.' : @PORT
L.PROC - > ,, ' PH
L.PROC - > ,, ' H' : @VALUE , >
L.PROC - > ,, ' STON
*
MAX ,, DCOUNT @VALUE, @VM)
IF MAX > ) THEN
  FOR NDX ,, 2 TO MAX
    L.PROC - > ,, ' H' : @VALUE , NDX > : ' '
  NEXT NDX
END
*
L.PROC - > ,, ' HSAVE.LIST WRK.' : @PORT : ' '
L.PROC - > ,, ' PH
*
WRITE L.PROC ON ' MD ', ' PROC.' : @PORT
EXEC ' > : PROC.' : @PORT
DELETE ' MD ', ' PROC.' : @PORT
  
```



SEL.PRINT.DEST	Prompt for a report destination
<p>This paragraph is typically called from the Output Redir Process slot on the F6-Params F6-Addit screen of a report. It replaces the normal "Output to Screen, Printer..." and provides greater control over the output options for an individual report.</p>	
<p>The single parameter used in this process is an undelimited string of characters which define the options that are appropriate for this given report. If the report should output to:</p>	
Screen	Use "S";
Printer	Use "P";
Aux Printer	Use "X";
DIF	Use "D";
File	Use "F";
Background	Use "B";
HTML	Use "H"
<p>Or any combination of codes, such as SPF for Screen, Print, and File.</p>	
<p><i>Note: Later versions of SB+ have a standard OUTPUT.SELECT process that can be used instead to provide the same functionality.</i></p>	

```

LOCAL L.OPTS,L.VALID.OPTS,L.NDX,L.MAX,L.MSG,L.ADD.COMMA,L.BUTTONS,L.RTN.FLAGS
*
L.OPTS          ,, @PARAM
L.VALID.OPTS    ,, ' S,P,X,D,F,B,H'
L.VALID.OPTS 2> ,, ' Screen,Print,Aux,DIF,File,Bgnd,HTML'
*
L.MAX          ,, DCOUNT L.VALID.OPTS  >,' ,' )
L.NDX          ,,
L.ADD.COMMA    ,, 0
*
L.BUTTONS     ,, ''
L.RTN.FLAGS   ,, ''
*
WHILE L.NDX   L.MAX DO
  IF INDEX L.OPTS,FIELD L.VALID.OPTS  >,' ,' ,L.NDX), ) THEN
    IF L.ADD.COMMA THEN
      L.BUTTONS     ,, L.BUTTONS : ','
      L.RTN.FLAGS   ,, L.RTN.FLAGS : ','
    END
  END
*
L.BUTTONS     ,, L.BUTTONS : FIELD L.VALID.OPTS 2>,' ,' ,L.NDX)
  
```

Precision Solutions, Inc.
Standard Implementation Processes

Last Updated: 19 Aug 2009



```
L.RTN.FLAGS ,, L.RTN.FLAGS : FIELD L.VALID.OPTS >,' ',L.NDX)
L.ADD.COMMA ,,
END
*
L.NDX ,, L.NDX +
REPEAT
*
IF L.BUTTONS ,, '' THEN
  L.BUTTONS ,, L.VALID.OPTS 2>
  L.RTN.FLAGS ,, L.VALID.OPTS >
END
*
L.BUTTONS ,, L.BUTTONS : ',Exit'
L.RTN.FLAGS ,, L.RTN.FLAGS : ', '
*
L.MSG ,, ' Select Report Destination\ ' : L.BUTTONS : ' \ ' : L.RTN.FLAGS
DISP 4,L.MSG
*
IF @RTN.FLAG ,, THEN
  EXIT
END ELSE
  @VALUE ,, @RTN.FLAG
  EXIT 0
END
```



SYSTEM49	Return Unidata subroutine call stack
<p>This /PD.B process and associated subroutine will parse the SYSTEM(49) [subroutine call stack] from Unidata and will return it in @VALUE as a last-in/first-out stack attribute-delimited stack similar to @PROC.NAME. This then can be used to determine which subroutines have been called en route to the current process.</p> <p style="text-align: right;">10/12/2007</p>	

```

SUBROUTINE SYSTEM49
*
* Written By: Kevin King
* Date:      2 Oct 2007
* Project:   Standard
* Description: This routine will return the list of subroutines (stripping
*             everything but the subroutine name) from SYSTEM_49).
*
*****
* Modifications
*****
* Date..... Changed By..... Description of Change.....
*
*****
* Include_s)
*****
$INCLUDE DMSKELCODE COMMON
*
CALL STACK ,, SYSTEM_49)
STACK.CNT ,, DCOUNT_CALL.STACK,@AM)
*
VALUE ,, ''
*
FOR STACK.LOOP ,, TO STACK.CNT
  SUBR.INFO ,, CALL.STACK STACK.LOOP,2)
  CONVERT ' \ ' TO ' / ' IN SUBR.INFO
  SUBR.NAME ,, FIELD SUBR.INFO,' / ' ,DCOUNT SUBR.INFO,' / ' ) [2,9999]
  IF SUBR.NAME NE '' ) THEN
    INS SUBR.NAME BEFORE VALUE >
  END
NEXT STACK.LOOP
*
RETURN
    
```



UNIQUE	Verify a value entered into a multivalued list is unique.
<p>This /PD.V (Validation) Process can be called from another process to verify that the entry into a multivalued list is unique. This allows the unique-check validation code to be embedded into a validation paragraph or subroutine instead of having to be listed in the field definition.</p> <p>When this process is used, it verifies that the value in @VALUE is not in the multivalued list as identified by the current prompt (details in @LINE). If the value is in the list, an error message is displayed and @RTN.FLAG is set to 1. Note that if braces are used following the U: code (i.e. U: {,}), no error message will be displayed though the @RTN.FLAG common variable will be set on error.</p> <p>(And yes, this is not a misprint. The validation code is only two characters.)</p> <p style="text-align: right;">08/20/2004</p>	

U:



VAL.TABLE	Validate the entry is in a SB+ Code Table
<p>VAL.TABLE is a validation paragraph that can be used in place of the SB+ generated validation process and verifies that the user's entry is in the table as named. Using this process (with SEL.TABLE, a /PD.S), you can achieve the same results as what SB+ generates for you when you press F3 on the conversion code. The benefits to this approach include:</p> <ul style="list-style-type: none">• You don't have to remember what the original conversion was to restore it after the F3 on the Conversion prompt;• The field definition is markedly smaller because the complex intuitive help process is no longer used. (SEL.TABLE would be used instead).• It's easier to include table validations as a part of a more complex validation using this process instead of the E: expression validation approach. <p>From a validation slot, use:</p> <p>C:VAL.TABLE, <i>tablename</i></p> <p>...or...</p> <p>C:VAL.TABLE, <i>tablefile</i>, <i>tablename</i></p> <p>...where <i>tablename</i> is the name of the table to validate against, and <i>tablefile</i> is the name of the file where that table is stored, if not the <i>sysDEFN</i> file.</p>	

```
IF TABLE (@PARAM) ,, '' THEN
  ERROR "[E42]"
  EXIT
END
```