

Download Definition

Custom Software Created by



Documentation Prepared By:

Kevin King
Precision Solutions, Inc.
303/651-7050

Last Updated:
3/9/2007 9:12:00 PM

Table of Contents

Table of Contents	2
Introduction	3
Starting the Download Definition Tool	3
Exiting the Download Definition Tool	3
The Main Screen	4
Download ID	4
Description	4
Unix Directory	4
Unix File Name	4
Unix File Format	5
Dictionary File Name	5
Data File (if diff)	5
Use Heading (Y/N)	6
Selection Criteria	6
Sort Fields	6
Comments	6
Expressions	6
Width	7
Just	7
The F4-Del Key	7
The F5-Addit'l Screen	8
Proc At Start	8
Proc Before Select	8
Proc After Select	9
Proc Aft Read	9
Proc At End	9
Special Format Proc	9
Max Rows Per File	9
The F7-Copy Screen	10
Exporting to Excel	11
Incorporating a Download Into Other SB+ Programming	11
The Expression Language	12
Expression Basics	12
Literals	12
Field Names	12
Common Variables	12
Shortcuts for Common Variables	13
Next Generated Number	13
System Variables	13
Attribute and Value References	13
Substring Extraction	14
Conversions	14
Expression Operators	14
Expression Functions	16
Mathematical Functions	16
String Functions	17
Logical Functions	19
I/O Functions	19
Miscellaneous Functions	20

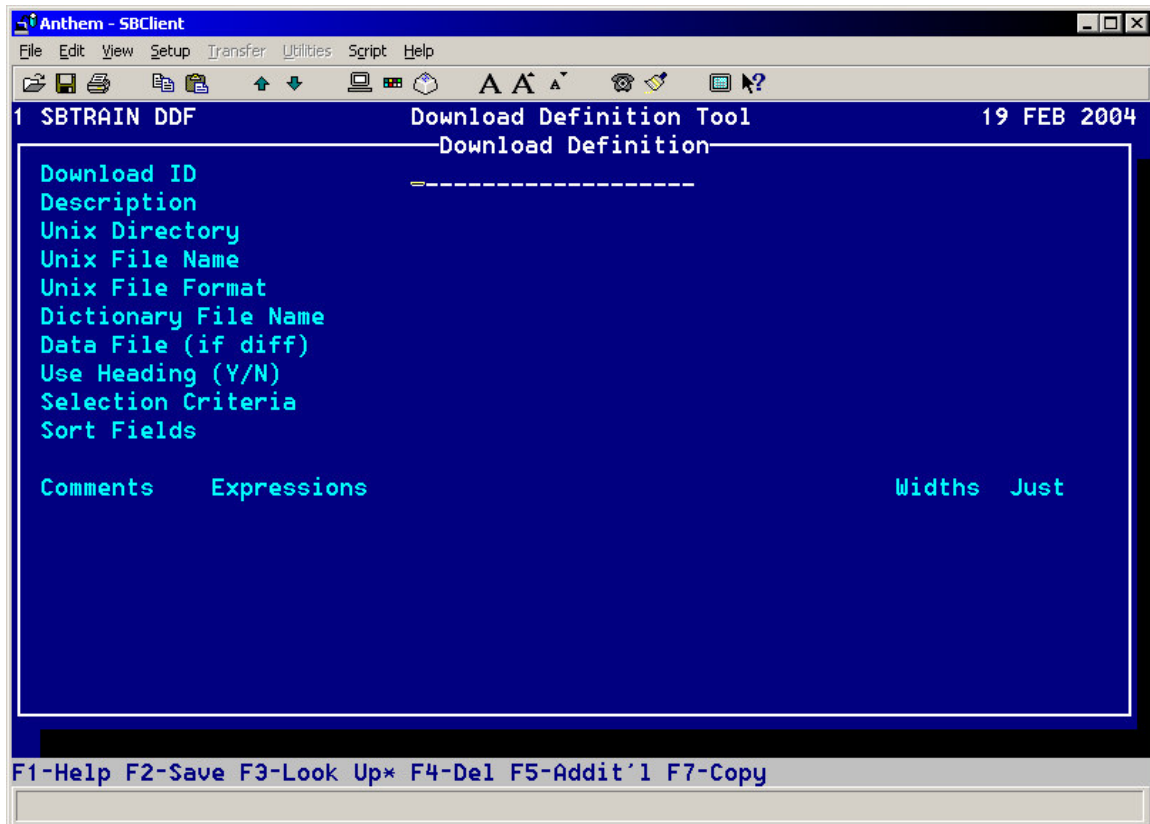
Introduction

Download Definition is a custom SB+ tool for extracting information from an SB+ application. This extracted information can be then used in other applications that run external to the business system, such as other databases or PC applications like Excel.

Fitting seamlessly into the SB+ application, Download Definition provides developers the ability to rapidly and effortlessly build files in a variety of formats. From tab separated values or commas, vertical bars to any format you can imagine, the download definition tool provides the foundations for getting your information out of your application in the right format and with the least amount of effort or investment in development time.

Starting the Download Definition Tool

The Download Definition tool can be started with a slash command at any menu or input prompt in SB+. Simply enter /DDF.DEFN to start the tool. When started, the main tool screen appears as follows:



Using this tool, you will create records (*download definitions*) that define the information you want exported, the format of that exported information, and any special processing you might want to do with that exported information (like moving it to Excel). Everything about the export is defined by the information entered into the prompts in this tool.)

Exiting the Download Definition Tool

To exit the tool, press [Escape] or <cr> when the cursor is on the Download ID prompt.

The Main Screen

The following paragraphs discuss the prompts on this main screen:

Download ID

Every download needs a name. When you create a download, the tool will create a process for you using the same name. Therefore, you cannot create a download that has the name of a previously defined process. When the download is complete and you press F2 to save, you can then immediately enter *ldownloadname* (where *downloadname* is the name of your download) to invoke your download. Alternatively, you can also incorporate download definitions into other larger programs as if they were any normal SB+ process.

To recall an existing definition, press F3 at this prompt and follow the prompts to select from a list of previously defined definitions.

Description

The description simply describes the download definition in your own terms and is, for the most part, a comment entry. When you press F3 on the Download ID prompt, the description entered here will be shown in the selection window, so it's recommended to enter something useful for assisting you in finding the definition at some future moment in time.

Unix Directory

At this prompt, enter the name of a file in the current environment that points to a directory in the underlying operating system. Don't be confused by the "Unix" in the name, it simply represents any underlying operating system, whether Unix, HP-UX, AIX, or Windows.

It's important to note that this tool requires a file in the local account that points to a directory in the underlying operating system, such as a DIR-type file in Unidata or a Super-Q in D3.

Unix File Name

Enter the name of the export file to be created in the directory entered previously. For example, if you want to export to a file called "export.tsv", simply enter that name (without quotes) at this prompt.

Use care in applying extensions (characters following a dot in the name) with your files, ensuring the extension matches the format of the data. For example, here are a couple of common formats and extensions:

Format Description	Extension	Download Type
Tab Separated Value	.tsv .txt	T
Comma Separated Value	.csv	C
System Data Format (ASCII SDF)	.txt	S

Of course, depending on the application that will be using the exported information, you may have other rules to consider when naming the output file.

Sometimes the output file name cannot be determined during design time, but must instead be calculated at run time. For example, what if the date is to be included in the output file name? This is obviously something that cannot be set in the design of the download, but rather must be calculated each time the download is invoked. To accommodate this need, you may enter any SB+ expression enclosed in

parentheses for the file name at this prompt. For example, to create an output file called ORDERmmddyyy, the following expression might be used for the file name:

```
("ORDER": (OCONV (@DATE ( ) , "D4" ) "MCN" ) )
```

See chapter 8 of *SB+ Solutions* for more information about building and understanding SB+ expressions.

Unix File Format

This prompt defines the output format for the exported information. Press F3 at this prompt to see the different formats that are available. These formats include:

ASCII SDF (System Data Format)	The SDF format is perhaps the oldest textual export format available. It is a fixed width format consisting of columns of information with specific widths and justifications per column, with a carriage return at the end of each line.
Vertical Bar Format	The Vertical Bar Format is a dynamic format that places a vertical bar between data elements, with a carriage return at the end of each line.
CSV (Comma Separated Value)	The CSV format is like the Vertical Bar Format, but it uses a comma between data values. Any data value that includes a comma will automatically be quoted.
SB+ Process Will Format	This format does not do any actual formatting of the output, but rather calls an SB+ process to do the actual output formatting. (See F5-Addit'l for more information about this process.)
Tab Separated Value	The TSV (Tab separated value) is like the Vertical Bar Format except that tabs (ASCII 9) are used as the delimiter. No data values are automatically quoted with this format. This is generally the preferred format for moving information to PC applications.

Dictionary File Name

Later on this screen there is a prompt entitled "Expressions" where standard SB+ expressions (that define the columns of exported information) can be entered. If any of these expressions refer to fields in a file (by field name), the name of that file should be entered at this prompt.

Like most SB+ tools, this is a required entry. You must have a primary file for your download, and in most cases this is also the primary dictionary file.

Data File (if diff)

If the file where the fields live is different from the file where the data lives, enter the name of the file where the data lives into this prompt. If the fields and data both live in the same file, this entry can be left blank.

Sometimes the data file name cannot be determined during design time, but must instead be calculated at run time. For example, what if the date is to be included in the data file name? This is obviously something that cannot be set in the design of the download, but rather must be calculated each time the download is invoked. To accommodate this need, you may enter any SB+ expression enclosed in parentheses for the file name at this prompt. For example, to use a data file named LEDGERyyyy, the following expression might be used for the data file name:

```
("LEDGER": (@DATE "D4Y" ) )
```

See chapter 8 of *SB+ Solutions* for more information about building and understanding *SB+* expressions.

Use Heading (Y/N)

If the exported information is to have a heading row, enter "Y" at this prompt. Otherwise enter "N". See the "Comments" prompt below for more information about column headers.

Selection Criteria

At this prompt, enter the selection criteria for the exported information. This defines which records in the data file will be selected for export. All of the standard *SB+* selection criteria options are available here, so any selection criteria you might normally use in an *SB+* report, selection process, or periodic update can be expressed here.

See chapter 8 of *SB+ Solutions* for more information about the options available for *SB+* selection criteria

Prelude Users: Use (@PARMS (39)) in Selection Criteria to selection from a URM screen.

Sort Fields

If the information to be exported is to be in any specific order, enter the field names to sort on in this prompt. To cause a sort to be in descending order, append a "D" to the end of that particular sort field. For example, to sort by an ascending date and by a descending city name, the following sort fields might be used:

```
DATE CITY(D)
```

If any of these fields are multivalued, the result will be an "exploded sort", causing each value of a multivalued set to appear on a row by itself. Single valued fields will be repeated for each value of the multivalued field. Therefore, this is very useful for normalizing multivalued sets of information.

Comments

This prompt begins the definition of the actual columns of exported information. For each entry in this prompt, there must be a corresponding entry in the Expressions prompt. Depending on the format being exported, you may also have an entry in the Widths and Justs columns as well for each entry shown here.

If "Use Heading" is "Y", this column defines the text of the heading for this column. If "Use Heading" is "N", then the information entered here is truly just a comment for your reference.

Sometimes a column heading cannot be determined at design time, because the value of that heading might change each time the information is exported. To accommodate this need, you may enter any *SB+* expression enclosed in parentheses at this prompt. That value will then be calculated at run time and the resulting answer used as the column heading. For example, if the heading for a given column were actually saved in the common variable @USERDATA(1)<5,2>, the following could be used for the comment entry:

```
(@USERDATA(1) <5, 2>)
```

See chapter 8 of *SB+ Solutions* for more information about building and understanding *SB+* expressions.

Of course, if headings are turned off for the report, entering an expression into this prompt will have no affect on the output of the exported information.

Expressions

For each comment/heading entry, enter a valid SB+ expression to be used for outputting this column for each selected record. In some cases, this may be as simple as a field name from the dictionary named earlier. In other cases, this may involve using the F(.) function to read from another file, or possibly even the P(.) function to call a process to calculate this column's value. Any valid SB+ expression can be entered here.

The common variable @OTHER(3) contains the number of the row being output. If you're building expressions for Excel that need to refer to other columns of information, you can use this to determine the row that is currently being processed.

See chapter 8 of SB+ Solutions for more information about building and understanding SB+ expressions.

Width

If the information is being exported in ASCII SDF format, enter the width of this column into this prompt.

Just

If the information is being exported in ASCII SDF format, enter one of the following codes to define the justification and fill character for this column:

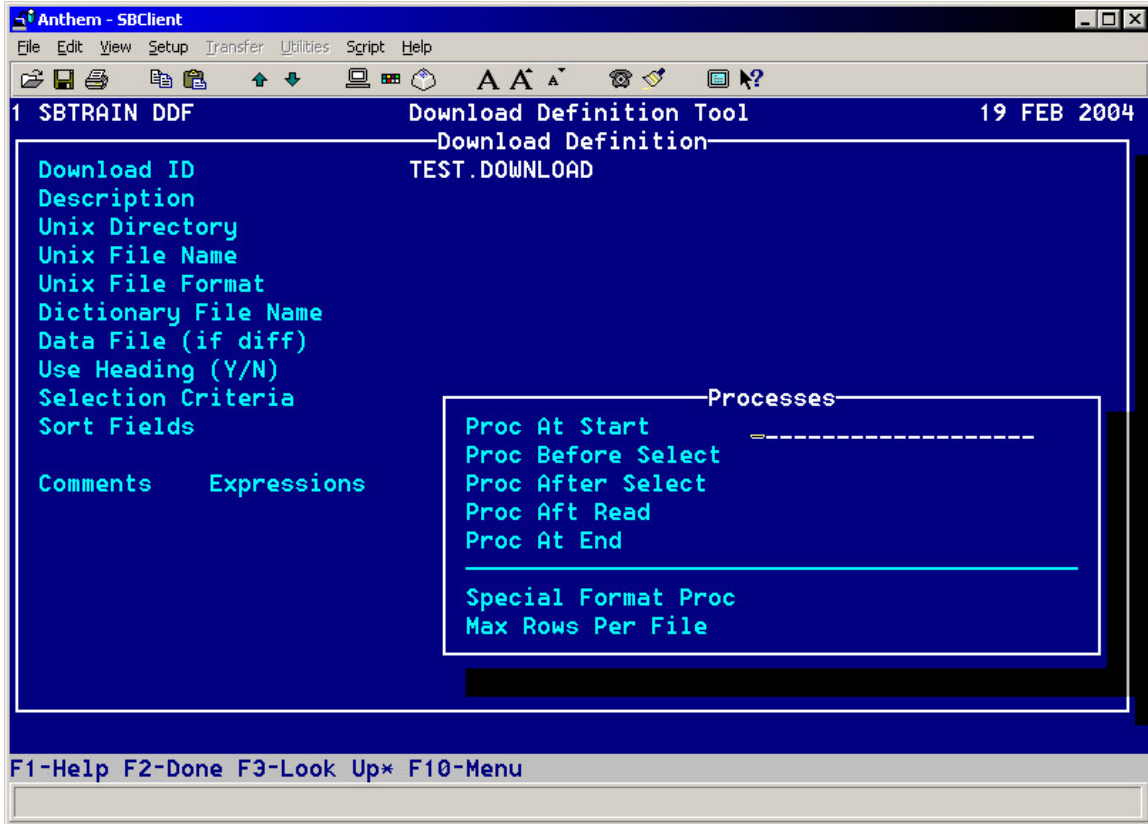
Code	Description
LS	Left Justified, fill with spaces
LZ	Left Justified, fill with zeroes
RS	Right Justified, fill with spaces
RZ	Right Justified, fill with zeroes

The F4-Del Key

Use the F4-Del key to delete the current definition. When this key is pressed, the message "Are You Sure You Wish To Delete (Y/N)". Enter "Y" to delete the definition, or "N" to keep the definition intact.

The F5-Addit'l Screen

Additional information is available from the F5-Addit'l key. When this key is pressed, the following screen appears:



Proc At Start

In this slot, enter the name of a process to be called at the beginning of the download, before any information has been exported. This is useful for setting up column headings into memory variables (like @USERDATA(...)) or for setting work variables that may be needed throughout the execution of the program.

To have a heading exported before any rows of information, use this process to set @PARMS(20)<1> to "HEAD" and @PARMS(20)<2> to a multivalued list of heading lines to be exported at the start of your output file.

This process is called only once at the beginning of the download. If @RTN.FLAG is set to "1" in this process, the download will terminate.

Proc Before Select

In this slot, enter the name of a process to be called before records have been selected. This then allows you to modify any selection criteria you may have in memory before it is actually used by the download processor. Furthermore, you may also use this process to configure an export heading as described in the Proc At Start.

This process is called only once immediately prior to any records being selected for export. If @RTN.FLAG is set to "1" in this process, the download will terminate.

Proc After Select

In this slot, enter the name of a process to be called after records have been selected. This process could be used to clear working values, much like the previous slots, and can also be used to configure the export heading as described in Proc At Start.

This process is called only once immediately after records have been selected for export. At this point there is an active select list so if you do anything in this slot that uses that list, the export will have nothing queued for output. Take care to avoid exhausting the active list with any processes called from this slot. If @RTN.FLAG is set to "1" in this process, the download will terminate.

Proc Aft Read

As each record is read for export, any process listed in this slot will be called. In this process you can modify the information in the current record and key (in @RECORD and @KEY, respectively) and then use these possibly changed values in the expression. To cause a particular record to be skipped over in the output, simply set @RTN.FLAG to 1 in this process, and that single record will be skipped. This can be a useful technique for implementing selections that may be difficult or impossible with selection criteria.

Unless your process specifically writes the record, you can modify the record or key without any fear of disturbing the information as it exists in your files. Once the information has been extracted from the record and output to the export file, the current record and key will be discarded by the download tool.

This process is called once for each record selected.

Proc At End

After all of the selected records have been exported, this process (if present) will be called. This is useful for performing whatever cleanup or transfer of the information you may need. For example, the "TO.EXCEL,*file,item*" process can be listed in this slot, and when invoked it will transfer the named item to the local PC¹ and open the file in Excel.

This process is called only once at the end of the download.

Special Format Proc

This process is used when the format list (in Unix File Format) is "P" (SB+ Process Will Format). This process slot allows you to export information in whatever format you can build in an SB+ process.

When this format is used, the download tool builds a dynamic array of export information (based on the expressions listed earlier) into a multi-attributed @RECORD. (Each column of exported information becomes a single attribute in that record.) This process is then called just before the information is output to the export file. In the process, you can then reformat the contents of the exported information into the common variable @VALUE. Whatever is in @VALUE at the end of your process is what will be output to the export file. Note that one input record may result in several output records; simply create several attributes in @VALUE and each attribute will become a row of the export file.

Max Rows Per File

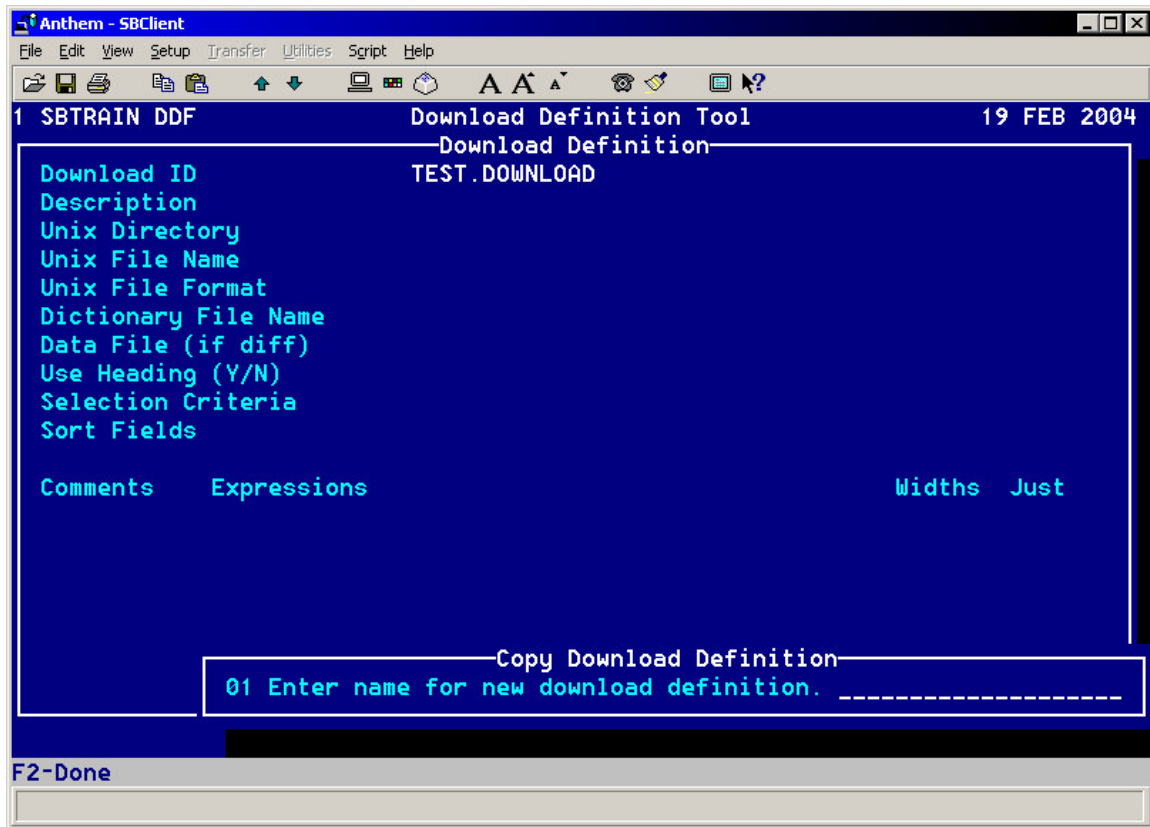
¹ Using SBClient only.

Some applications have restrictions on the amount of information that can be imported in a single file. (For example, Excel can import only 65536 rows per file.) If the information being exported has such a limitation, enter the maximum number of rows for any given file in this prompt.

During the execution of the download, if more rows than expressed here are output, a new file will be created with an extension of ".1". If that file grows larger than the number expressed here, a new file will be created with an extension of ".2". This will continue until all rows have been output.

The F7-Copy Screen

From the main screen, press F7-Copy to copy the current definition to a new name. When this key is pressed, the following screen is displayed:



At this prompt, simply enter the name for the copy of this definition. Once this has been entered, the definition will be copied. **You will still be in the original definition following the copy.**

To rename the current definition, simply copy the definition to another name and then delete the original using F4-Del.

Exporting to Excel

In the Proc At End slot, you may enter the following process to cause the exported information to be transferred to the local PC² and opened in Excel:

```
TO.EXCEL,filename,itemname
```

In this process call, *filename* is the name as listed in the Unix Directory prompt, and *itemname* is the name listed in the Unix File Name prompt. If the Unix File Name prompt uses an expression to calculate the name, you *may* need to call another process to calculate the name to be passed to the TO.EXCEL process. However, if the only variable part of the name is a port number (which is commonly done), you can incorporate the literal ":PORT " in the *itemname* and that part will be replaced at run time with the port number, such as:

```
TO.EXCEL,EXPORTS,WRK:PORT
```

So if this were used to create an export in EXPORTS called WRK.5 (for port 5), this would cause WRK.5 (and any secondary files, based on max item size) to be transferred to the local PC and opened in Excel.

Incorporating a Download Into Other SB+ Programming

Once created, a download becomes a regular SB+ process (viewable with the /PD.B tool) that can be incorporated into any other SB+ programming. Simply invoke the process by name at the appropriate spot in your application to start the export. If the download definition uses run-time parameters in the selection criteria (such as NAME = "?"), those prompts will appear with the download is invoked, much like it would if this were an SB+ report.

And herein lies the secret to incorporating downloads into other SB+ programming – think of the download as just another kind of report. Whatever you might do to incorporate an SB+ ReportWriter report into your programming, do the same for the download!

Prelude Users: The Prelude software contains a user report manager (/URM) that allows you to define prompts for building selection criteria. This Prelude tool integrates very easily into the download definition with just a little extra programming (not typically included with this tool). If you would like to integrate these two tools, contact Precision Solutions for assistance.

² SBClient only.

The Expression Language³

The expression language forms the basis for many features of SB+. Expressions are the root of SB+ conversions and derived values, field defaults, E: validations, and also are a very important part of the paragraph language.

The following paragraphs take an inside look at constructing expressions, the different parts of the expression language, and the myriad of functions available in the language.

Expression Basics

Literals

The most fundamental form of expression is a literal, which can be either a number or alphabetic string. If alphabetic string, the literal must be enclosed in either single or double quotes (as long as the beginning and ending quote match). For example, the following are legal literals:

```
56
'Hello, World'
"SB+"
```

Field Names

In place of a literal, a field name may be referenced. If we have a CUSTOMER file with fields named NAME, ADDRESS, and CITY, we can reference any of these fields by name, such as:

```
NAME
ADDRESS
CITY
```

Of course, the case of the field name must match the case of the field definition name. Therefore, if the above are valid field names, the following are invalid references to the fields:

```
Name
Address
City
```

Any time a multivalued field is referenced by name, the value as pointed to by the common variable @CNT is used. To reference the multivalued field as a whole you must either set @CNT to 0 (which can be very dangerous), or by using the POS(...) function, as follows:

```
@RECORD<POS(NAME)>
```

Common Variables

Another fundamental component of the expression language is references to common variables. Each of the common variables are referenced with a preceding "@" sign, which makes common variables visually distinct from other types of variables. For example, the following are valid common variable references:

```
@RECORD
@KEY
@ORIG.REC
@ACTION
```

³ Reprinted from SB+ Solutions, chapter 8.

Shortcuts for Common Variables

Any attribute in @WORK can be referenced by a shortcut in the form W_{n,m} (where n is an attribute number, and m is an optional value reference). Therefore, to reference @WORK<5>, W5 could be used. To reference value 2 of attribute 4 of @WORK, W4,2 could be used.

The same is true for @OTHER.REC, except the shortcut letter is "O" (not zero) instead of "W". To reference @OTHER.REC <2>, O2 could be used. To reference the third value of attribute 7, O7,3 could be used.

Lastly, when referencing specific attributes of @RECORD, the variable name can be omitted. Therefore, attribute 5 of @RECORD can be referenced as:

@RECORD<5>

...or more simply:

<5>

Similarly, value 2 of attribute 5 can be referenced as <5,2>, instead of @RECORD<5,2>.

Next Generated Number

Another fundamental component of the expression language is the "G" command which assigns next sequential numbers. The syntax of this expression element is as follows:

G_{n,m,sysid}

In this syntax, the "n" parameter determines the accumulator to generate the number from. The "m" parameter is an optional length parameter, and if supplied it tells SB+ to right justify the assigned number in a field of zeros, "m" characters in length.

If the accumulator is to be read from another system record, the other system ID can be supplied in the "sysid" parameter.

System Variables

There are a few variables that are referenced the same way as common variables (i.e. they have a preceding "@" character), but aren't really variables at all. These include:

@AM or @FM - The attribute mark character

@VM - The value mark character

@DATE - The current date (input converted format)

@TIME - The current time (input converted format)

Attribute and Value References

A literal or common variable may be followed by an attribute and/or value reference, in the form:

<attribute{,value}>

For example, to reference attribute 5 of @OTHER.REC, the following could be used:

@OTHER.REC<5>

To reference value 2 of attribute 4 of @RECORD, the following could be used:

```
@RECORD<4,2>
```

If the value reference is included, but is zero, it is assumed that the value reference has been omitted. Therefore, the following are equivalent:

```
@PARMS(2)<4,0>  
@PARMS(2)<4>
```

Substring Extraction

Following a common variable, literal, or field name (and the optional attribute and value reference), substring extraction may be defined, in the form:

```
[starting character,length]
```

In this syntax, the first parameter defines the first character to be extracted and the second parameter defines the number of characters to be extracted. For example, to extract the second character of the third value of attribute 5 of @RECORD, the following could be used:

```
@RECORD<5,3>[2,1]
```

...or, using shortcuts the following is equivalent:

```
<5,3>[2,1]
```

Conversions

Following an expression, an input or output conversion may be specified (in double quotes). This allows you to do conversions on a field without using OCONV or ICONV. Honestly, these are carryovers from earlier versions of SB+ from the time when ICONV and OCONV were not supported. Still, they are a valid option in the expression language.

For example, to output-convert @VALUE using the "MR2" conversion, the following could be used:

```
@VALUE"MR2"
```

Or, to input-convert @VALUE using the same conversion:

```
@VALUE"MR2[I"
```

Note the "[I" on the end of the conversion. This is how SB+ determines whether to input-convert or output-convert the value being converted. (The "[" is simply an open bracket, not a control character.)

Expression Operators

Simple expressions can be combined with a variety of different operators. These variables are divided into five levels of precedence, as follows:

Level 1 (highest precedence, strongest binding)

- * Multiplication
- / Division (with rounding)
- | Division (without rounding)

Level 2

+ Addition
- Subtraction

Level 3

: Concatenation

Level 4

= Equal To
Not Equal To
< Less Than
<= Less Than or Equal To
> Greater Than
>= Greater Than or Equal To

Level 5

AND Logical AND: If both operands are true, the result is true (generally "1"). If one or the other operand is not true (i.e. zero), the result is zero.
OR Logical OR: If one operand or the other is true, the result is true. If both operands are zero, the result is zero.

(The lower the level, the tighter the binding between operators and operands.)

Based on this list of operators and their corresponding precedence levels, the following are valid expressions:

PRICE * 1.5
3 + 2 * 5

Precedence is important particularly in this last example. In this example, the correct result is 13, not 25. Of course, parentheses can always be used to change the way that a particular expression is processed, as in the following example:

(3 + 2) * 5

...in which case the result of 25 is correct.

When running interpretively, SB+ will "sense" multivalued fields, and apply the operators on a value-by-value level when adding, subtracting, or multiplying multivalued fields. In generated code, however, SB+ is unable to sense this. Therefore, in the 2.x release, always do everything at a single-value level. In the 3.x release, SB+ has added the MV(...) and SV(...) functions which can help you get the proper results.

Expression Functions

(In the following examples, the keyword "exprn" is used to signify any valid SB+ expression of any complexity.)

Mathematical Functions

ABS(exprn)	This function will return the absolute value of the expression enclosed in quotes. ABS(@VALUE) - If @VALUE = -3, the function will return 3.
INT(exprn)	This function will return the integer part of the expression enclosed in quotes. Any remainder is discarded. INT(@RECORD<5>) - If @RECORD<5> = 4.6, the function will return 4.
MATH(expression 1, operator, expression 2)	This function will call a subroutine called STRING.MATH as catalogued in the current account, passing the expressions and operator as parameters. For more information about how this is used, see the comments in the program named STRING.MATH in DMSKELCODE.
MOD(value, divisor)	This function will return the remainder of the first parameter divided by the second. MOD(PRICE,PCT) - If price is 10 and PCT is 3, the function will return 1.
NOT(exprn)	This function will return the logical opposite of the expression enclosed in quotes. For example, if the expression is true, NOT(exprn) will return false, and vice versa. IF NOT(PRESENT) THEN - If PRESENT is true, NOT(PRESENT) will return false. If PRESENT is false, NOT(PRESENT) will return true.
S(exprn)	This function will take a multivalued expression and will return the sum of the multivalues. If referencing a multivalued field name in the expression, use caution. Under certain circumstances, this function doesn't work properly because of the implied multivalued position that is inherent in referencing multivalued fields by name. However, if you reference a multivalued field using the POS(...) function, this problem can be avoided. @VALUE = S(AMTS) - If AMTS is 21]5]6, @VALUE will be assigned the value 32.

String Functions

AT(column, row)	<p>This function will return the cursor positioning codes for placing the cursor at the column and row as listed in the parentheses. This is equivalent to the BASIC function @(column,row)</p> <p>PRINT AT(20,10) - Cursor will be moved to column 20, row 10 on the screen. Note that this is not applicable to a GUI screen.</p>
CHAR(exprn)	<p>This function will return the ASCII character for the number or expression in parentheses.</p> <p>CHAR(65) - The character "A"</p>
COL1()	<p>Following a FIELD(...) function, this function will return the character position of the character immediately preceding the substring returned by the FIELD(...) function. Note that there are no parameters to this function.</p>
COL2()	<p>Following a FIELD(...) function, this function will return the character position of the character immediately following the substring returned by the FIELD(...) function. Note that there are no parameters to this function.</p>
DCOUNT(exprn, delimiter)	<p>This function will return the number of values in the expression as delimited by the delimiter.</p> <p>NDX = DCOUNT(@VALUE,@VM) - If @VALUE = 21]5]6, the DCOUNT(...) function will return 3, because there are 3 values in the string.</p>
DEL(exprn, attribute, value)	<p>This function will return the expression with the attribute and/or value removed. If the value parameter is zero, an attribute will be removed. If the value parameter is greater than zero, a value will be removed. If the value parameter is less than zero, any amount of weirdness could occur, depending on your operating environment.</p> <p>XYZ = "21]6]5" @VALUE = DEL(XYZ,1,2)</p> <p>@VALUE will equal "21]5" after this code.</p>
DUP(string1, string2)	<p>This function will count the number of values in the second parameter, and will duplicate the first parameter to have as many values. For example, to add "5" to all of the values in a multivalued field called LIMIT, the following could be used:</p> <p>DUP(5,LIMIT) + LIMIT</p> <p>The DUP(...) function will return a multivalued list of 5's with as many values as there are values in LIMIT. The addition will then be done on a value-by-value level.</p>
FIELD(exprn, delimiter, instance)	<p>This function will return a substring of the first parameter using the second parameter as a delimiter. The third parameter determines which substring to extract. For example, if @VALUE contains:</p> <p>24*13*2*12</p> <p>...the following will return the number "2":</p> <p>FIELD(@VALUE,'*',3)</p>
INDEX(exprn to search, exprn to find, instance)	<p>This function will search the first parameter for a specific instance of the second parameter, and will return the character position where the search string was found, or "0" if not found. For example, if @VALUE contains:</p> <p>24*13*2*12</p> <p>...the following will return the value "9":</p> <p>INDEX(@VALUE,'1',2)</p>

<p>INS(exprn,attribute,value,value to insert)</p>	<p>This function will return the expression with the value to insert inserted at the attribute and value position. If the value position is "0", an attribute will be inserted. If the value position is greater than "0", a value will be inserted into the attribute before the position as named.</p> <p>The "value" parameter may also be one of the following alphabetic literals, enclosed in quotes:</p> <p>AR Insert the value into the string in ascending right justified order. DR Insert the value into the string in descending right justified order. AL Insert the value into the string in ascending left justified order. DL Insert the value into the string in descending left justified order.</p> <p>@VALUE = 21 : @VM : 3 @VALUE = INS(@VALUE,1,-1,"X") ;* @VALUE will equal 21]3]X</p> <p>@VALUE = 21 : @VM : 3 @VALUE = INS(@VALUE,1,1,"X") ;* @VALUE will equal X]21]3</p> <p>@VALUE = "ABC" : @VM: "DEF" @VALUE = INS(@VALUE,1,"AL","BIG") ;* @VALUE will equal ABC]BIG]DEF</p>
<p>LEN(exprn)</p>	<p>This function will return the number of characters in the expression.</p> <p>@VALUE = 21 NDX = LEN(@VALUE) ;* NDX will equal 2</p>
<p>LOC(find string, search string, delimiter)</p>	<p>This function will search through the second parameter for the find string in the first parameter, based on the delimiter in the third parameter to separate values. An exact match must be found, otherwise this function will return zero.</p> <p>@VALUE = 21 : @VM : 4 : @VM : 5 NDX = LOC("4",@VALUE,@VM) ;* NDX will equal 2 (4 is in the second position in the multivalued list)</p> <p>@VALUE = '21,4,5' NDX = LOC(5,@VALUE,',') ;* NDX will equal 3 (5 is the third position in the comma-delimited list)</p>
<p>MATCHES(exprn, pattern)</p>	<p>This function will determine if the expression matches the pattern. If it does, the function returns "1". If not, the function returns "0".</p> <p>MATCHES(@VALUE,"3N1X") - Will return true (1) if @VALUE is three numeric digits followed by one character of any type.</p>
<p>SEQ(exprn)</p>	<p>This function will return the ASCII numeric code for the first character of the expression. For example:</p> <p>SEQ('A')</p> <p>...will return the value 65.</p>
<p>REPL(find string, replacement string, search string)</p>	<p>This function will return the search string with the find string replaced by the replacement string.</p> <p>REPL("G","X","GEORGE") - Will return XEORXE.</p>
<p>TABLE(exprn)</p>	<p>This function will lookup the value of @VALUE in the table as named by the expression, and will return the description of the code (or null if the code is not found). The expression may return either a single table name, or the name of a table file and table name separated by a comma, as follows:</p> <ul style="list-style-type: none"> • TABLE("tablename") -Lookup @VALUE in the table named "tablename" in the current XXXDEFN file and return the description. • TABLE("file,tablename") - Lookup @VALUE in the table named "tablename" in the file as named and return the description.

TRIM(exprn)	This function will return the expression with extraneous spaces removed. Leading and trailing spaces are trimmed off, as are duplicate spaces between words. @VALUE = TRIM(" THIS IS PADDED ");* @VALUE will = "THIS IS PADDED"
-------------	--

Logical Functions

ALPHA(exprn)	If the expression in parentheses is strictly alphabetic, this function will return "1". Otherwise, this function returns "0". ALPHA(21) - Will return false (0). ALPHA("XYZ") - Will return true (1). ALPHA("X@") - Will return false. ALPHA("X2") - Will return false.
CASE(condition 1, value 1;condition 2, value 2;...)	This function will evaluate the first condition and if it is true, return the first value. If this first condition is false, the second condition is evaluated, etc., until a condition is true, or all conditions have been evaluated. It is very important to always include an exception case so the CASE(...) function will always return some value.
IF(condition, true value, false value)	This function will evaluate the condition in the first parameter. If the condition is true, the true value is returned. If the condition is false, the false value is returned. IF(@VALUE=4,21,6) - Will return 21 if @VALUE is 4. Otherwise, this will return 6.
NUM(expression)	If the expression in parentheses is strictly numeric, this function will return "1". Otherwise, this function returns "0". Note that this function typically sees a preceding minus and the first period as numeric characters. A minus anywhere else in the string, or more than one period will cause this test to return "0". (This can vary depending on your operating environment.)

I/O Functions

Truth be told, there is only one I/O function. However, there are enough variations on this function to make it seem like several different functions. The following illustrates these variations:

F(file, record ID)<attribute>	This function will perform a READV of the file, record, and attribute as listed and will return the value as read. F("CUSTOMERS",@VALUE)<4> - Will use @VALUE as a key to a record in the CUSTOMERS file, and will return attribute 4 of that record. The record is discarded after the value has been extracted.
F(file, record ID)	This function will read the record from the file as listed into @OTHER.REC and will return the entire record to the expression. F("CUSTOMERS",@VALUE)- Will use @VALUE as a key to a record in the CUSTOMERS file, and will return the entire record to the expression, leaving the record in @OTHER.REC.
(F(file, record ID))<attribute>	This variation on the F(...) function is a combination of the previous two forms with all of the benefit of both. When this syntax is used, the entire record is read into @OTHER.REC, however, only the attribute as listed is returned to the expression. (F("CUSTOMERS",@VALUE))<4> - Will use @VALUE as a key to a record in the CUSTOMERS file, and will read the entire record into @OTHER.REC. Then, attribute 4 is extracted and returned. @OTHER.REC remains intact with the record after the expression has completed.

Miscellaneous Functions

B(exprn)	<p>This function will call a subroutine as named by the expression, returning @VALUE back to the expression. If the expression has a comma in it, everything up to the comma is the subroutine name and everything following the comma will be moved into the common variable @PARAM for passing the parameters to the subroutine.</p> <p>@PARMS(2) = B("SUBNAME") - Will call a subroutine named "SUBNAME". The value returned from the subroutine (in @VALUE) will then be copied to @PARMS(2).</p>
DECRYPT(exprn)	<p>This function works with the ENCRYPT(...) function. An expression encrypted using ENCRYPT(...) can be decrypted and returned to its normal state using this function. In this syntax, exprn is the expression to be decrypted. The decrypted value will be returned to the left side of the equation, as in:</p> <p>@RECORD<1> = DECRYPT(@RECORD<1>)</p>
ENCRYPT(exprn)	<p>This function works with the DECRYPT(...) function. An expression is passed as the parameter, and the result is the value in an encrypted format. This is a great feature for storing values (like credit card information) in a format that cannot be easily viewed using the editor or other unprotected system tools.</p> <p>@RECORD<1> = ENCRYPT(@RECORD<1>)</p>
ICONV(exprn, conversion)	<p>This function will perform an input conversion on the expression using the conversion as listed. For example, this function can be used to convert a date from human-readable format into its internal form, such as:</p> <p>ICONV("12.31.96","D")</p>
MV(exprn)	<p>This function does not calculate any value. Instead, it tells SB+ that the expression should be treated as a multivalued expression, and each individual multivalued value should be extracted and handled separately. For example, if the following lines are in a paragraph:</p> <pre>W4 = 1 : @VM : 2 W5 = 3 : @VM : 4 @VALUE = MV(W4 + W5)</pre> <p>...the result stored in @VALUE will be 4]6, because instead of adding the complete W4 to the complete W5, the MV(...) function told SB+ to handle and calculate the addition for each multivalued value separately. This function is only required when SB+ could otherwise not figure out whether a field was multivalued -- such as a work field referenced by Wn, a @PARMS(...) or @USERDATA(...) element, or a local variable. When referencing fields by name (based on field definitions), SB+ has enough information to know whether the field is multivalued or single valued.</p> <p>When running paragraphs interpretively, SB+ can usually figure out whether an expression element is a single or multivalued expression, and acts accordingly. However, when code is generated with /GC, SB+ occasionally needs you to use this function (or the SV(...) function) to explicitly declare your intent.</p>
OCONV(exprn, conversion)	<p>This function will perform an output conversion on the expression using the conversion as listed. For example, this function can be used to convert a date from internal form to a human-readable format, such as:</p> <p>OCONV(10000,'D2')</p> <p>...will return the value "18 MAY 95".</p>
P(process name)	<p>This function will call a process as named by the expression, returning @VALUE back to the expression. If the expression has a comma in it, everything up to the comma is the process name and everything following the comma will be moved into the common variable @PARAM for passing the parameters to the process.</p> <p>@PARMS(2) = P('PROCNAME') - Will call a process named "PROCNAME". The value returned</p>

	from the process (in @VALUE) will then be copied to @PARMS(2).
POS(field name)	<p>This function will return the field position (attribute number) of the field as named in the parameter. This is not a real-time function. When the expression is evaluated (which is usually when the derived value, conversion, or default expression is entered, or the paragraph is saved) the POS(...) function returns a value and this value is placed into the expression. In other words, if the field position changes after it has been referenced in a POS(...) function, the expression will still look at the old position until it is regenerated. This can be done either by pressing <cr> through and resaving the field definition, or by using the Regenerate Expressions tool in SB+.</p>
SV(exprn)	<p>This function does not calculate any value. Instead, it tells SB+ that the expression should be treated as a single value expression, and each individual multivalued (if present) should NOT be extracted and handled separately. For example, if the following lines are in a paragraph:</p> <pre> W4 = 1 : @VM : 2 W5 = 3 : @VM : 4 @VALUE = SV(W4 + W5) </pre> <p>When the paragraph is running interpretively, the result stored in @VALUE will be 4]6 (same as the MV(...) function would return). When code is generated for the paragraph, the generated code will produce a non-numeric error message, because it's trying to add two values which are not purely numeric. Note that like the MV(...) function, this function is only required when SB+ could otherwise not figure out whether a field was single valued -- such as a work field referenced by <i>Wn</i>, a @PARMS(...) or @USERDATA(...) element, or a local variable. When referencing fields by name (based on field definitions), SB+ has enough information to know whether the field is multivalued or single valued.</p> <p>When running paragraphs interpretively, SB+ can usually figure out whether an expression element is a single or multivalued expression, and acts accordingly. However, when code is generated with /GC, SB+ occasionally needs you to use this function (or the MV(...) function) to explicitly declare your intent.</p>