APPENDIX

2

Advanced User Topics

Introduction

This appendix contains more details about some advanced topics such as data set options, processing inserts and deletions, using BY keys, multiple views, multiple windows, and how you can optimize your selection criteria. The discussions supplement other portions of this manual.

Data Set Options

Data set options allow you to override some of the run-time options that are stored in view descriptors. Two options are currently available: S2KPW= and S2KMODE=. You can use these as data set options on SAS PROC statements.

The options specified with the DATA= argument on a PROC statement override those stored in a view descriptor.

Description

S2KPW=password
: allows you to override the SYSTEM 2000 password stored in the view descriptor. If no password is stored in the view descriptor, the S2KPW= option must be used to provide a valid password for the database.

  The password must be an alphanumeric value one to four characters long with no embedded blanks, optionally enclosed in single quotes. Passwords longer than four characters will be truncated with a warning message. If the password is a special character, it must be a single character (that is, a one-character password) enclosed in single quotes.

  S2KPW= is an option on the DATA= argument where DATA= specifies a SYSTEM 2000 view descriptor that will be used as input to a SAS procedure other than the DBLOAD procedure. (Passwords specified in the DBLOAD procedure cannot be overridden.)

S2KMODE | S2KMD=S | M
: allows you to override the SYSTEM 2000 access mode stored in the view descriptor. M means the database files are allocated in a region controlled by the Multi-User software. S means you are executing the procedure as a single-user job, where you allocate the database files in your job and execute a separate copy of SYSTEM 2000 software.

  S2KMODE= is an option for the DATA= argument where DATA= specifies a SYSTEM 2000 view descriptor that will be used as input to a SAS procedure other than the DBLOAD procedure. (The DBLOAD procedure uses the mode specified for a new database or stored in the view descriptor for an incremental load.)

  S2KMD= is an alias for S2KMODE=.

Data Set Option Syntax

You can specify the S2KPW= and S2KMODE= data set options with the DATA= argument on any PROC statement except PROC DBLOAD. They are effective for that single execution of the procedure. Data set options will override the corresponding values stored in the view descriptor.

The following example executes the FSEDIT procedure using a view descriptor named EMPPOS. The data set options specified in the PROC statement will execute SYSTEM 2000 software in single-user mode, using the password DEMO.

```
proc fsedit data=vlib.emppos
  (s2kmode=s s2kpw=demo);
run;
```

Using Multiple View Descriptors or Multiple Windows

You can use more than one view descriptor in a single SAS session, but only one can be open for updating. This restriction applies to either one window that opens two view descriptors or two windows that each open one view descriptor. You cannot have the QUEST procedure and a SAS procedure or a DATA step that refers to a SYSTEM 2000 view descriptor active at the same time in two windows, unless one is single-user mode and the other is Multi-User mode.

Deleting Data Records

If you are deleting an observation, for example, with the FSEDIT procedure, use the DELETE command. Note, however, that the system sets all the values of items in the view descriptor (that is, not the unselected items in the same record) to missing and *removes* the lowest level data record from the database if at least one of its items was selected for display. Ancestor records are also removed if they do not have other descendant records. The data records must be locked, and they are not removed until you move to a different observation.

The DELETE command does not remove items or records unless your password has U-authority for the appropriate items and records.

Inserting Data Records

Insertion of database records occurs as a result either of update operations from various SAS procedures or of database loading from the DBLOAD procedure. When you insert a new observation, it can cause the insertion of more than one SYSTEM 2000 database record. The number of inserts depends on how many levels are in the database and on a comparison between the data being inserted and the data in the last observation read, if any. During an insert operation, levels having data different from the prior observation, if any, result in a SYSTEM 2000 database record insertion.

You can perform inserts with SYSTEM 2000 software in two modes:

- insert mode
- optimized load mode.

For the DBLOAD procedure, you determine the mode with the S2KLOAD statement. You must use insert mode if you are loading new records into existing logical entries.

Either load mode can be used for the DBLOAD procedure when you are loading entire logical entries. Optimized load mode is more efficient than insert mode, but it has some restrictions:

- Data must be sorted in data tree order prior to the load.
- Entire logical entries are always inserted.
- The number of inserts and the levels at which inserts are performed depend on the order of the data and on what fields change from observation to observation.
- Your input cannot be a SYSTEM 2000 view in the same environment.

Note: During optimized load processing, your output database will be open in exclusive use mode with rollback disabled temporarily. △

Insert mode is suitable for mass insertion of descendant records into existing logical entries with the DBLOAD procedure. Similar to load mode, the engine determines where the new records go, depending on the values of fields in the observation. When you insert an observation, the engine compares it to the prior observation. Depending on how many fields have changed, one or more records are inserted at the levels that have changed. Also, you can specify a BY key to help determine the point of insertion. BY keys are discussed in the next section.

Using a BY Key to Resolve Ambiguous Inserts

Each time the interface view engine is called to add an observation, it inspects the changes you made from observation to observation, in order to determine how many data records to insert into the database. The purpose is to reduce data redundancy.

If none of the data changed, or if the changes were only at the bottom level of the view, the engine needs to insert only a single new data record at the bottom level. Since the engine inserts at least one record for any add, and since only one record is called for here, there is no question about how many records to insert for this kind of add. That is, the insert is not ambiguous.

However, if any data values changed in records above the bottom record in the path, an ambiguous situation develops. A certain number of new records seem to be required by your changes, but some of the new data may already be present in existing database records. That is, the actual number of new records to be added to the database may be different.

In optimized load mode, the engine ignores the ambiguity; it inserts all of the new data that are at or below the highest level record that changed. Therefore, when you specify optimized load mode, be sure that your incoming data are always sorted by major-to-minor sort keys at every level (from level 0 down to the bottom level in the view). If the data are not sorted correctly, redundancy will not be eliminated.

On the other hand, in insert mode, the engine can determine whether some of the new data are already present in an existing record. If so, the engine needs to insert records only for those data not already present in the database. If not, the engine needs to insert a record at every level that changed. Use the optional BY key capability to eliminate redundancy and to help the interface view engine find an existing path for inserting the new records.

A BY key is similar to a BY group in the SAS System, which groups observations based on one or more fields. Many SAS procedures process records in BY groups. Also, some updates in the DATA step are performed by matching specified BY variables in different data sets. A similar matching process occurs with BY key items in the SAS/ACCESS interface to SYSTEM 2000 software.

If you specify a BY key, it should contain one or more database items at each level above the bottom level in your view descriptor.

BY keys do cause extra processing time because the engine issues one or more where-clauses to look for already-existing records.

Examples of Using a BY Key

Suppose you have a view with C1 and C11 in the BY key and three observations.

```
C1        A         B
          |         |
C11      CCC       DDD
         / \        |
C21     1   2       3

      obs1  obs2   obs3
```

Suppose you are in the FSEDIT procedure on observation 1. You enter the DUP command and values A, CCC, and 4. This is not an ambiguous insert; a BY key is not required. The changes in values from observation 1 to your new input are confined to the bottom of the view. Here is the result.

```
C1        A         B
          |         |
C11      CCC       DDD
         /|\        |
C21     1 4 2       3
```

Now suppose you are in the FSEDIT procedure on observation 1. You enter an ADD or a DUP command and the values B, DDD, and 5 for C1, C11, and C21, respectively. The insert is ambiguous because all the fields in the new observation are different from observation 1. Without a BY key, the result is

```
C1        A         B         B
          |         |         |
C11      CCC       DDD       DDD
         /|\        |         |
C21     1 4 2       3         5
```

With a BY key, the engine finds the BY key values C1=B and C11=DDD in the database. The result then is

```
C1        A           B
          |           |
C11      CCC         DDD
         /|\         / \
C21     1 4 2       3   5
```

BY Key Considerations

The recommended way to use BY keys is as follows:

- include an item from each level above the bottom of the view
- standardize all database updates through the same view or consistent views.

Consider the following caution areas when you do not use BY keys in this way:

- *Potentially inadequate BY keys.* The engine does not enforce the important suggestion that a BY key must contain at least one item at every level above the bottom level in the view descriptor. However, if the BY key does not contain enough unique items, it may be inadequate to help the engine; the engine may behave as though there were no BY key at all.

- *Inconsistent use of a BY key.* The engine does not enforce consistent use of BY keys; one view descriptor might have a BY key and another might not. In this case, redundant data could be added to the database through the view descriptor that does not have a BY key. Also, some applications could enter redundant data through the QUEST procedure, which does not call the engine for database updates.

  If data are added in any way other than through a view descriptor using a BY key, the engine might find several qualified database records that match the incoming data. The engine would simply pick one record that works and use it when inserting the new records. Thus, the incoming data might be attached beneath a different existing record than the one you expect.

  To avoid this situation, make sure that all users who update the database follow the same rules. That is, ensure that all data entry is done through the interface view engine and that all users use the same view descriptor (or at least consistent view descriptors).

In addition, the notion of a prior observation is important during inserts, because the engine compares your new data to it. The prior observation is obvious for SAS procedures that pass through a file sequentially, for example, the DBLOAD procedure. However, other SAS procedures can jump around within a file at random, for example, the FSEDIT procedure.

When you add observations through procedures that do not use sequential processing, remember that the prior observation is the last observation that the procedure showed you. For example, in the FSVIEW procedure, the prior observation is the last observation that the procedure showed you at the bottom of your display before your first update.

In some cases, there is no prior observation, such as when you enter the DBLOAD procedure. That is, the procedure calls the engine to add an observation without any prior retrieval. If this situation occurs, the engine issues a GET1 ... LAST command for the record at the top of the view and retrieves the last record that was inserted in the database.

These details are not very important if you are using a BY key consistently.

Missing Values (Nulls)

SYSTEM 2000 software and the SAS System handle missing values (nulls) differently. The interface view engine that stands between the two systems must handle the differences in a predictable, useful way. The following topics discuss how missing values and empty records are handled in retrieval, update, and where-clause processing.

Retrieving Nulls

When the interface view engine is reading database records and constructing an observation, it could find missing data in the path of the data records that represent the observation. In a SYSTEM 2000 database,

- missing structure means the data record at the top of the view exists, but some or all of its descendant records do not exist.
- missing values means that the value for one or more items in a data record is null. Nulls for all item types consist of all binary zeros in the database.

In the SAS System, missing values in character variables are represented by all blanks. Missing numeric values are represented by a special floating point number.

When the interface view engine retrieves a null from the database, it sets it to be a missing value in the corresponding SAS observation. Because SYSTEM 2000 software preserves all blanks for TEXT and UNDEFINED values, a value containing all blanks for one of these item types would be interpreted as a missing value by a SAS procedure.

Updating Nulls

The interface view engine supports four kinds of updates: ADD, UPDATE, DUP, and DELETE.

ADD means adding an observation, which can have missing values. The interface view engine converts the SAS observation into a set of one or more SYSTEM 2000 data records, comprising the path defined by the view descriptor. Each variable in each record is converted from the SAS internal format to the SYSTEM 2000 format. Note that even if all variables in a SYSTEM 2000 record have missing values, the record will still be inserted into the database. That is, the complete path of data records is always inserted; lower level data records might contain all missing values.

UPDATE means updating an observation with a set of values, which can have missing values. If the observation being updated has no missing structure, each variable is converted from its SAS form into a SYSTEM 2000 form.

If the observation being updated has a missing structure in the database, the records that exist in the path will be updated with whatever values have changed since the path was retrieved. Missing structures will be inserted only if the values are not null.

DUP means duplicating the selected observation in the database, which could mean duplication of more than one database record.

DELETE means deleting an observation, which could mean deletion of more than one database record. For more information about deletions, see “Deleting Data Records” on page 129. Because SYSTEM 2000 software preserves all blanks for TEXT and UNDEFINED values, a value containing all blanks for one of these item types would be interpreted as a missing value by a SAS procedure.

Missing Values in Selection Criteria

SYSTEM 2000 software and the SAS System treat missing values (nulls) differently when processing where-clause conditions. SYSTEM 2000 software assumes that a null is outside the domain of values for an item. Therefore, the only way to qualify a null is by using the FAILS operator; the NE (not equal) operator will not qualify nulls. Also, a null will never satisfy a condition containing the LT (less than) or LE (less than or equal) operator. For example, if item C2 is null in some data records, the following item-to-item condition will never qualify those records, regardless of the respective values:

```
WHERE C1* > C2*
```

In fact, for any relational operator in an item-to-item condition, SYSTEM 2000 software never qualifies a record in which either of the items is null. Even if the condition is C1* = C2* and both items are null, the record will not qualify. In contrast, the SAS System assumes null fields (missing values) are equal to each other. Missing values have the following attributes in the SAS System:

- For numeric variables, a missing value is the lowest (most negative) possible value. It is lower than every other value that is not null.
- For character variables, a value of all blanks is considered to be a missing value.

When the SAS System processes a condition such as C1 >= C2, the qualified records would include every record in which C2 is missing, regardless of the value of C1. Also, the condition C1 = C2 would qualify records that have missing values for both C1 and C2, along with records where C1 and C2 have equal values that are not null.

Because of these different treatments, it can be important to know whether the SAS System or SYSTEM 2000 software is processing a particular portion of a where-clause. Where-clause processing is discussed in “Using a SAS WHERE Clause for Selection Criteria” on page 134. Briefly, the where-clause in a view descriptor is never seen by the SAS System and will be processed by SYSTEM 2000 software. However, the WHERE clause associated with the SAS procedure, the DATA step, or a SELECT statement in the SQL procedure can be processed partly by both the SAS System and SYSTEM 2000 software, depending upon whether individual conditions are meaningful to SYSTEM 2000 software.

Since missing values are different, a condition in a SAS WHERE clause that explicitly uses the dot (.) notation will never be seen by SYSTEM 2000 software. The SAS System will perform the qualification for such conditions. For more information about where-clause processing, read the next sections.

Using a SAS WHERE Clause for Selection Criteria

In addition to or instead of including a SYSTEM 2000 where-clause in your view descriptor for selection criteria, you can also specify a SAS WHERE clause in a SAS program for selection criteria.

Note: Unlike a SYSTEM 2000 where-clause stored in a view descriptor, a SAS WHERE clause is restricted to variables corresponding to items included in the view descriptor. (A SYSTEM 2000 where-clause can reference items contained in a view descriptor and items contained in the access descriptor that the view descriptor is based on.) △

When you specify a SAS WHERE clause, the SAS/ACCESS interface view engine translates those conditions into SYSTEM 2000 conditions. Then, if the view descriptor includes a SYSTEM 2000 where-clause, the interface view engine connects the conditions with the Boolean operator AND. By default, the SAS WHERE clause conditions are connected to the end of the view descriptor conditions. For example, if a view descriptor includes the condition

```
sex=female
```

and the SAS WHERE clause condition translates into

```
position=marketing
```

the resulting selection criteria are

```
sex=female and position=marketing
```

You can control the connection of the translated SAS WHERE clause and the SYSTEM 2000 where-clause conditions by including a connecting string in a SYSTEM 2000 where-clause included in a view descriptor. A connecting string indicates where you want the connection to occur. For example, suppose you have included the following SYSTEM 2000 where-clause in a view descriptor. (*SASAND* is one of the available connecting strings.)

```
*sasand* department=marketing
```

You then issue a SAS procedure including a SAS WHERE clause that produces the following condition:

```
salary gt 1000
```

The resulting selection criteria are as follows:

```
salary gt 1000 and department=marketing
```

For more information and examples on the available connecting strings, see "Connecting Strings" on page 139.

When the interface view engine translates SAS WHERE clause conditions into SYSTEM 2000 conditions, there are SAS WHERE clause capabilities that are not available in SYSTEM 2000 software. Therefore, it is possible to issue a SAS WHERE clause that cannot be totally satisfied by SYSTEM 2000 software.

To allow for this possibility, the interface view engine first evaluates the SAS WHERE clause and determines whether the conditions can be handled. The interface view engine may be able to partially satisfy a SAS WHERE clause, as in the following example:

```
proc print data=vlib.emp1;
where lastname < 'KAP'
  and payrate > 30 * overtime;
run;
```

The interface view engine translates as much of the SAS WHERE clause as possible, without producing incorrect results or a syntax error from SYSTEM 2000 software. In the previous example, SYSTEM 2000 software has no problem with the first condition, but the arithmetic in the second condition is not supported. The interface view engine uses the condition **`where lastname < 'KAP'`** to filter out as many data records as possible to improve performance. The conditions that are not supported are bypassed by the interface view engine, and post-processing (handled automatically by the SAS System) will be required after SYSTEM 2000 software does its subsetting. The engine bypasses

- unacceptable conditions.
- conditions connected with OR to unacceptable conditions.
- conditions that exceed a SYSTEM 2000 where-clause 1000-byte limit. If the SAS WHERE clause exceeds 1000 bytes, the rightmost portion of it is bypassed by SYSTEM 2000 software.

  When the interface view engine examines the SAS WHERE clause, it determines which conditions SYSTEM 2000 software can support. At this point, the engine has not processed the view descriptor where-clause. Later, when the engine processes the view descriptor where-clause, the possibility arises that the combined length of the SAS WHERE clause conditions acceptable to SYSTEM 2000 software and the view descriptor where-clause conditions could exceed 1000 bytes.

  If the engine has determined that SYSTEM 2000 software completely supports the SAS WHERE clause, and then it determines that the conditions cannot be combined due to the 1000 byte limit, an unrecoverable error occurs. To the procedure or DATA step, it appears as though the first "read" observation failed. You may need to carefully examine the error messages on the log to determine what actually happened.

In the following table,, assume C114 is a component in the bottom record of a view descriptor. (Remember that if there is no SYSTEM 2000 where-clause included in the view descriptor and no SAS WHERE clause specified in the SAS program, the interface view engine issues a default where-clause in the form of WHERE C*n* EXISTS OR C*n* FAILS, where C*n* is a component in the bottom record in the view descriptor.)

View Where-Clause	SAS WHERE Clause	SYSTEM 2000 Translation	Post-Processing Required?
C1=A	C2=B OR C3>C4+10	(C1=A)	Yes
C1=A	C2=B & C3>C4+10	(C1=A) & (C2=B)	Yes
C1=A	C2=B OR C3>C4	(C1=A) & (C2=B OR C3*>C4*)	No
C1=A	C2=B & C3	(C1=A) & (C2=B)	Yes
—	—	C114 EXISTS OR C114 FAILS	No
—	C3*20 < C5	C114 EXISTS OR C114 FAILS	Yes
—	C3 = C5	C3* = C5*	No

SAS WHERE Clause Conditions Acceptable to SYSTEM 2000 Software

The following information explains how the interface view engine translates acceptable SAS WHERE clause conditions into SYSTEM 2000 where-clause conditions.

□ The following operators are translated as indicated:

WHERE Clause Syntax	SYSTEM 2000 Translation
=	=
>	>
<	<
<>	!=
>=	>=
<=	<=
IS NULL	FAILS
IS NOT NULL	EXISTS
(	(
)	)
AND	AND
OR	OR

□ The interface view engine also translates BETWEEN and IN conditions, item-to-item comparisons, and the date format:

WHERE Clause Syntax	SYSTEM 2000 Translation
C1 BETWEEN 1 AND 3	C1 = 1*3
C1 IN (4,9,14)	C1=4 OR C1=9 OR C1=14

WHERE Clause Syntax	SYSTEM 2000 Translation
C4 > C5	C4* > C5*
C4 = '02AUG87'D	C4 = 08/02/1987

- SYSTEM 2000 software can handle a limited subset of SAS WHERE clause pattern matching, with the following prerequisites:
  1. The pattern must be less than 100 characters long.
  2. The pattern must have % as the rightmost character.
  3. Underscores are permitted only in leftmost position(s).
  4. The pattern cannot have % anywhere but in the leftmost or rightmost position.
  5. The pattern must have some characters that are not % or _.

Where Clause Syntax	SYSTEM 2000 Translation
C1 LIKE %ABC%	C1 CONTAINS ABC
C1 LIKE ABC%	C1 CONTAINS ABC IN 1
C1 LIKE _ABC%	C1 CONTAINS ABC IN 2
C1 LIKE __ABC%	C1 CONTAINS ABC IN 3

SAS WHERE Clause Conditions Not Acceptable to SYSTEM 2000 Software

Here is a list of some (but not all) SAS WHERE clause conditions that are not acceptable to SYSTEM 2000 software; they are handled automatically by SAS post processing.

- arithmetic expressions, for example,

```
WHERE C1 = C4 * 3
WHERE C4 < -C5
```

- expressions in which a variable or combination of variables assumes a value of 1 or 0 to signify true or false, for example,

```
WHERE C1
WHERE (C1 = C2) * 20
```

- concatenation of character variables
- truncated comparison, for example,

```
    C1 =: ABC
```

- DATETIME and TIME formats, for example,

```
'12:00'T
'01JAN60:12:00'DT
```

- SOUNDEX
- HAVING, GROUP BY
- NOT CONTAINS

- references to '.' for numeric variables or ''for character variables. Use **`WHERE C1 IS NULL`** if that is what you mean, not **`WHERE C1 = .`** (The interface view engine can translate C1 IS NULL into C1 FAILS.)

The NOT Operator

The SAS WHERE clause NOT operator and the SYSTEM 2000 where-clause NOT operator do not function the same way. If you want NOT to have its SAS meaning, put it in the SAS WHERE clause. If you want NOT to have its SYSTEM 2000 software meaning, put it in the view descriptor where-clause.

If you specify NOT in a SAS WHERE clause, NOT is transformed by the SAS WHERE clause parser first; the interface view engine never sees the NOT operator. Consider the following examples:

SAS WHERE Clause	What the Engine Sees
WH NOT LASTNAME = 'Jones';	WH LASTNAME NE 'Jones';
WH NOT LASTNAME > 'Baker';	WH LASTNAME <= 'Baker';
WH NOT (LASTNAME = JONES AND HIREDATE > '02aug82'd);	WH LASTNAME NE 'Jones' OR HIREDATE <= '02aug82'd;

In SYSTEM 2000 software, however, the logical converse of **`wh not lastname = 'Jones'`** is not **`lastname ne Jones`**. Rather, the logical converse of **`wh not lastname = 'Jones'`** is **`wh lastname ne Jones or lastname fails`**. Before any relational operator can find a match for a value, the value must exist. One reason for this is that nulls are not contained in SYSTEM 2000 indexes, and processing an operator such as NE could be expensive if it were not confined to indexed values.

Deciding How to Specify Selection Criteria

Use the following guidelines to determine when to use a SAS WHERE clause and when to use a SYSTEM 2000 where-clause to specify selection criteria.

SYSTEM 2000 Where-Clause

Include a SYSTEM 2000 where-clause in your view descriptor when you want to

- restrict users of view descriptors to certain subsets of data.
- use SYSTEM 2000 syntax and functionality, such as component names, stored strings, HAS, AT, and the NON-KEY specification.
- qualify using a database item that is not in the view descriptor.
- ensure that null (missing) values are treated in the way SYSTEM 2000 expects. (The SYSTEM 2000 style of handling nulls differs from the SAS style in that SYSTEM 2000 software never regards nulls as equal to other values, even zero or blank.)
- use the SYSTEM 2000 functionality of the NOT operator. (The SYSTEM 2000 style of NOT processing differs from the SAS style in that SYSTEM 2000 includes null values in the answer, where the SAS System may or may not.)

- □ prevent users from sequentially passing the entire database. (The DBA can also set the SYSTEM 2000 option to DISABLE FULL PASSES as a way of preventing sequential processing.)

SAS WHERE Clause

Use a SAS WHERE clause when the previous guidelines do not apply and you want to

- □ have more run-time flexibility in subsetting data
- □ use SAS WHERE clause capabilities that SYSTEM 2000 software does not support, such as arithmetic expressions or truncated comparisons.

Connecting Strings

The order in which SYSTEM 2000 software processes conditions can affect which data records are selected. This consideration is most obvious when you include a SYSTEM 2000 where-clause in a view descriptor and then specify a SAS WHERE clause in a SAS program that uses the view descriptor. By default, the interface view engine connects the translated SAS WHERE clause conditions with the Boolean operator AND to the end of the SYSTEM 2000 where-clause conditions.

Therefore, to affect the order of the connected conditions, you can include a connecting string in a SYSTEM 2000 where-clause to tell the interface view engine how you want to connect the conditions. The following examples illustrate how the engine would connect the conditions:

Table A2.1

View Where-Clause	SAS WHERE Clause	Connected Conditions
C1 = A	C110 > 27	(C1 = A) & (C110 > 27)
SAS & C1 = A	C110 > 27	(C110 > 27) & C1 = A
C1 = ’A’ *ANDSAS*	C110 > 27	C1 = ’A’ AND (C110 > 27)

Note: Remember that the interface view engine translates only those SAS WHERE conditions it understands. △

Available Connecting Strings

The following list summarizes the available connecting strings that you can specify in a SYSTEM 2000 where-clause included in a view descriptor:

Table A2.2

String	Expands To
SAS	(*SAS-conditions*)
ANDSAS	AND (*SAS-conditions*)
SASAND	(*SAS-conditions*) AND
ANDNK	AND (NK (*SAS-conditions*))

String	Expands To
NKAND	(NK (*SAS-conditions*)) AND
*ANDAT(*n*)	AND ((*SAS-conditions*)AT *n*)
*ATAND(*n*)	((*SAS-conditions*) AT *n*) AND
*ANDHAS(*record*)	AND (*record* HAS (*SAS-conditions*))
*HASAND(*record*)	(*record* HAS (*SAS-conditions*))AND
*HASSAS(*record*)	(*record* HAS (*SAS-conditions*))
NKSAS	NK (*SAS-conditions*)
*SASAT(*n*)	(*SAS-conditions*)AT *n*

Syntax Limitations

You can specify a connecting string in a SYSTEM 2000 where-clause after a keyword or a special character. For example,

```
C1 = A AND *SAS*
```

is acceptable, but the following syntax is not:

```
C1 = A *ANDSAS*
```

To use the previous syntax, you can include a delimiter (special character), as shown below:

```
C1 = 'A' *ANDSAS*
```

Optional Omission of a SAS WHERE Clause

If a view descriptor includes a SYSTEM 2000 where-clause with a connecting string, and you do not execute a SAS WHERE clause, there will be nothing to substitute. For example, suppose you have included the following SYSTEM 2000 where-clause in a view descriptor:

```
C1 = A AND *SAS*
```

You issue a SAS program specifying a SAS WHERE clause that produces the following SYSTEM 2000 condition:

```
C110 > 27
```

If you do not specify a SAS WHERE clause in the SAS program, the "dangling connector" would result in a SYSTEM 2000 error.

```
C1 = A AND
```

If you want the flexibility of omitting the SAS WHERE clause, you can use the *ANDSAS* and *SASAND* connecting strings. For example, suppose you use *ANDSAS* instead of *SAS*:

```
C1 = 'A' *ANDSAS*
```

Then, even if you did not specify a SAS WHERE clause, there would not be a problem.

```
C1 = 'A'
```

Using OR

You cannot have an OR operator connecting a connecting string to other parts of a view descriptor where-clause. For example, the following view descriptor where-clauses are not acceptable:

```
C1 = A OR *SAS*
C1 = C OR (C1 = A OR C1 = B) *ANDSAS*
```

However, you can use OR in the following example:

```
(C1 = A OR C1 = B) AND *SAS*
```

Using HAS, AT, and NON-KEY

As mentioned, the HAS and AT operators and the NON-KEY specification are available in a SYSTEM 2000 where-clause, but they are not available in a SAS WHERE clause. With certain connecting strings, however, you can make their function more useful in the SYSTEM 2000 where-clause. Plus, you have the option of omitting the SAS WHERE clause without introducing errors or unexpected results. Here are a few examples:

Table A2.3

View Where-Clause	SAS WHERE Clause	Selection Criteria
C1='A' *ANDNK*	C2=B OR C3=X	C1='A' & (NK C2=B OR NK C3=X)
C1='A' *ANDNK*		C1='A'
C1='A' *ANDHAS(C0)	C21=B & C22=X	C1='A' AND (C0 HAS (C21=B & C22=X))
*ATAND(12) C1=A	C21=B	C21=B AT 12 & C1=A

SYSTEM 2000 Stored Strings

When you include a SYSTEM 2000 where-clause in a view descriptor, you can either use where-clause syntax as explained in “SYSTEM 2000 Where-Clause” on page 90, or you can refer to a SYSTEM 2000 stored string. A stored string is syntax contained in a SYSTEM 2000 database definition that can be invoked by using the string number or name. Either a complete where-clause or a portion of one can be stored. For example, you can store part of a SYSTEM 2000 where-clause in the database, such as

```
sex=female
```

Suppose you assign string number C1001 to the string. When you include a where-clause in a view descriptor, you can refer to the string number, for example,

```
department=marketing and *c1001*
```

When the selection criteria are processed by SYSTEM 2000 software against the database, the result is

```
department=marketing and sex=female
```

However, when the interface view engine confronts the view descriptor where-clause, the engine can check for errors only until it encounters the string reference. The engine cannot access the string definition at this point and therefore cannot expand the string to validate your syntax. Also, the engine cannot check the syntax that follows the string expansion; therefore, you must be more careful with the where-clause construction. However, the engine will append a SAS WHERE clause at the end of the view descriptor where-clause if this has not been done prior to the occurrence of a SYSTEM 2000 string reference.

Therefore, if you specify a stored string in a view where-clause, follow these rules in the where-clause syntax after the string reference:

- Use only valid SYSTEM 2000 item component names or numbers.
- Enter all keywords and any non-numeric values in uppercase.
- Avoid using any connecting string.
- Avoid using textual values containing significant blanks.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ACCESS® Interface to SYSTEM 2000® Data Management Software: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999.

SAS/ACCESS® Interface to SYSTEM 2000® Data Management Software: Reference, Version 8

ISBN 1–58025–549–3

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999