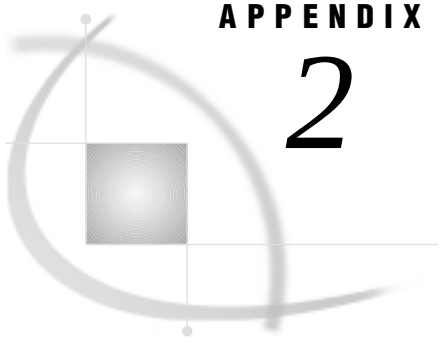

APPENDIX

2

Advanced User Topics

Introduction

This appendix contains details about some advanced topics such as using data set options, using multiple view descriptors, deleting and adding observations, using BY keys, and processing null values, as well as topics pertaining to selection criteria. The discussions supplement other portions of this book.

Data Set Options

In order for the ADABAS interface view engine to obtain ADABAS dictionary information, it needs certain ADABAS information. Specifically, the engine needs either a NATURAL DDM name or an ADABAS file number, in addition to a library identifier, a user identifier, passwords, cipher codes, and a database identifier.

If any of this information is required to access an ADABAS file or a NATURAL DDM but is not specified in the SAS/ACCESS view descriptor or cannot be obtained from either the ADBEUSE or ADBAUSE CSECT, you must use the appropriate data set option in your SAS procedure statement to supply the appropriate value.

Data set options allow you to specify these values. Data set options also allow you to override certain values that are specified in view descriptors but not enforced by Assign Security=YES.

Each data set option is an option in the DATA= specification where DATA= specifies a view descriptor that will be used as input to a SAS procedure. Data set options apply only for the duration of that procedure.

The following example executes the FSEDIT procedure using a view descriptor named VLIB.USAINV. The data set option specified in the PROC statement will execute ADABAS using the NATURAL SECURITY password INVOICE.

```
proc fsedit data=vlib.usainv (adbnatpw='invoice'};
run;
```

The available data set options appear below. Options marked with an asterisk (*) are enforced by Assign Security=YES. That is, if Assign Security=YES, the values specified in the view descriptor take precedence over values specified with a data set option; the data set option is ignored.

ADBCC='*cipher-code*'
specifies a cipher code for the target ADABAS file.

ADBDBID=*dbid*
specifies a database identifier for the target ADABAS file.

ADBFILE=*file-number*
specifies an ADABAS file number. The ADBFILE and ADBDDM data set options are mutually exclusive. If you specified a DDM name in the view descriptor, you can use ADBDDM, but you cannot use ADBFILE. If you specified an ADABAS file number instead, you can use ADBFILE but not ADBDDM.

ADBDEL=N|NO|Y|YES
allows you to override the default value for the interface view engine's systems option that determines whether a record containing periodic group fields should be completely deleted or its periodic group fields set to nulls. The default is set by the ADBDEL systems option in the ADBEUSE CSECT.
NO means set the fields to null; YES means delete the entire record.

ADBDDM='*ddm-name*'
specifies a NATURAL Data Definition Module (DDM) name. The ADBFILE and ADBDDM data set options are mutually exclusive. If you specified a DDM name in the view descriptor, you can use ADBDDM, but you cannot use ADBFILE. If you specified an ADABAS file number instead, you can use ADBFILE but not ADBDDM.

ADBL3=N|NO|Y|YES |O|ONLY
controls the use of the ADABAS L3 command by the interface view engine and what commands are used when L3 cannot be used. The L3 command optimizes WHERE and sort processing, with dramatic results for very large ADABAS files;

however, there are limitations on when the command can be used. See "Retrievals with Only a WHERE Clause" on page 108 for more information.

NO means the L3 command should not be used; YES means L3 is used and S1 and S9 are used if L3 cannot be used; ONLY means L3 is used and S2 is used, or an error is generated, when L3 cannot be used.

ADBNATAP='*library-id*' *
specifies a NATURAL SECURITY library identifier.

ADBNATPW='*password*' *
specifies a NATURAL SECURITY user password.

ADBNATUS='*user-id*' *
specifies a NATURAL SECURITY user identifier.

ADBPW='*password*' *
specifies an ADABAS password for the target ADABAS file.

ADBSECCC='*cipher-code*' *
specifies an ADABAS cipher code for the NATURAL SECURITY system file.

ADBSECDB=*dbid*
specifies an ADABAS database identifier for the NATURAL SECURITY system file.

ADBSECFL=*file-number*
specifies an ADABAS file number for the NATURAL SECURITY system file.

ADBSECPW='*password*' *
specifies an ADABAS password for the NATURAL SECURITY system file.

ADBSYSCC='*cipher-code*' *
specifies an ADABAS cipher code for the DDM system file.

ADBSYSDB=*dbid*
specifies an ADABAS database identifier for the DDM system file.

ADBSYSFL=*file-number*
specifies an ADABAS file number for the DDM system file.

ADBSYSPW='password' *
specifies an ADABAS password for the DDM system file.

ADBTRACE=*option*
specifies a trace option, which analyzes problems in SAS software. The default is ADBTRACE=0. If you specify ADBTRACE=1, WHERE clauses are displayed in the log. For more information on ADBTRACE, see "Debug Information for Problems" on page 115.

Using Multiple View Descriptors

You can use more than one view descriptor in a single SAS session, but only one can be open for updating. This is the default mode of operation.

For information on how to modify the engine to support multiple view descriptors in a single SAS session, contact your SAS Technical Support Representative.

Deleting an Observation

If the ADABAS file on which a view descriptor is based does not contain a periodic group, deleting an observation (for example, with the FSEDIT procedure DELETE command) causes a logical record to be deleted from the ADABAS data.

If the ADABAS file on which a view descriptor is based does contain a periodic group (the periodic group may or may not be included in the view descriptor), the results of deleting an observation depend on the status of the ADBDEL systems option, which is set either in the ADBEUSE CSECT (see "Systems Options for the ACCESS Procedure and the Interface View Engine" on page 115) or by a data set option (see "Data Set Options" on page 120).

- When ADBDEL=N (which is the default setting), the following results occur:
  - If there is only one periodic group occurrence (regardless of how many periodic group fields are in the view descriptor) and there are no other periodic group fields in the ADABAS file, deleting the observation containing the one occurrence causes the logical record containing the occurrence to be deleted.
  - If there are multiple occurrences for any periodic group field(s) in the view descriptor or if there are other periodic group fields in the ADABAS file, deleting the observation containing values from a periodic group occurrence causes the selected values for that occurrence to be set to null. The record is not deleted.
- If ADBDEL=P, the entire logical record is deleted, even if there are multiple occurrences for periodic group fields in the view descriptor or if there are other periodic group fields in the ADABAS file.

Adding an Observation

Adding ADABAS data as a result of update operations from various SAS procedures may cause the interface view engine to decide whether to add a new logical record to the ADABAS file or modify an existing logical record, for example, add an occurrence to a periodic group. The purpose of the engine making this determination is to reduce data redundancy.

The engine compares values in the new observation to be added to values in the previous observation. If the contents of the previous observation do not help determine whether to add or modify, a new logical record is added.

However, some of the new values may already reside in the ADABAS file, so a new record is not necessary. This situation occurs if a periodic group is included in a view descriptor, and the new data (which do not reside in the ADABAS file) occur only in variables corresponding to data fields that are part of that periodic group.

The interface view engine can determine whether this situation exists. If not, a new logical record can be added. If so, an existing record can be modified. The optional BY key specification makes this possible. See "Using a BY Key To Resolve Ambiguous Inserts" on page 122.

Using a BY Key To Resolve Ambiguous Inserts

When the interface view engine is called to examine additional ADABAS records in order to add a new periodic group occurrence, the engine must decide whether to add a new logical record or modify an existing one. The purpose is to reduce data redundancy.

You can help in the resolution of this decision by specifying a BY key. You can specify BY keys in the access descriptor by using the KEY statement. If Assign Names=NO, you can use the KEY statement to specify BY keys in the view descriptor. Only elementary data fields that are designated as ADABAS descriptors can be specified as BY keys.

A BY key is a set of match variables. A data field is a good candidate for a BY key if it uniquely identifies a logical record.

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 data fields in the SAS/ACCESS interface to ADABAS.

The BY key comparison process is as follows:

1. If values for a BY key match a record already in the ADABAS file, it will be modified. That is, the interface view engine inserts a new occurrence within a periodic group.
2. If values for a BY key do not match an existing record, a new record is added to the ADABAS file.

BY Key Examples

The following examples illustrate that using a BY key helps keep data organized and prevents unnecessary duplication of data.

Suppose you are working with the following two ADABAS logical records, which make up three SAS observations as shown in Output A2.1 on page 123. The data field named DF1 is specified as a BY key. DF2 is a periodic group consisting of data fields DF21 and DF22.

Output A2.1 By Key Example Containing Two ADABAS Logical Records of Three SAS Observations

Data Fields	DF1	DF2		
		DF21	DF22	
Record 1	A	CCC	1	(obs 1)
		CCC	2	(obs 2)
Record 2	B	DDD	3	(obs 3)

Example 1

You are in the FSEDIT procedure on observation 1. You enter an ADD or a DUP command and the values A, CCC, and 4. This is not an ambiguous insert, and a BY key is not required. Output A2.2 on page 123 shows the result.

Output A2.2 Results of Entering an ADD or DUP Command

Data Fields	DF1	DF2		
		DF21	DF22	
Record 1	A	CCC	1	(obs 1)
		CCC	2	(obs 2)
		CCC	4	(new observation (obs 4))
Record 2	B	DDD	3	(obs 3)

Example 2

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 data fields DF1, DF21, and DF22, respectively. This is an ambiguous insert because all the values you are entering are different than the ones in observation 1. If there were not a BY key, the result would be as shown in Output A2.3 on page 124.

Output A2.3 Results of an Ambiguous Insert

```
Data Fields        DF1            DF2
                                     DF21   DF22
Record 1           A                 CCC    1       (obs 1)
                                     CCC    2       (obs 2)
                                     CCC    4       (obs 3)
Record 2           B                 DDD    3       (obs 4)
Record 3           B                 DDD    5       (new observation)
```

With a BY key, the engine locates the BY key value DF1=B. Output A2.4 on page 124 shows the result.

Output A2.4 Results with a BY Key

```
Data Fields        DF1            DF2
                                     DF21   DF22
Record 1           A                 CCC    1       (obs 1)
                                     CCC    2       (obs 2)
                                     CCC    4       (obs 3)
Record 2           B                 DDD    3       (obs 4)
                                     DDD    5       (new observation)
```

Example 3

You are in the FSVIEW procedure, looking at the first three observations. You decide to add the values B, DDD, and 7 at the end. The current position is the third observation on the display. Output A2.5 on page 125 shows the result with no BY key.

Output A2.5 Results without a By Key

Data Fields	DF1	DF2		
		DF21	DF22	
Record 1	A	CCC	1	(obs 1)
		CCC	2	(obs 2)
		CCC	4	(obs 3)
Record 2	B	DDD	3	(obs 4)
		DDD	5	(obs 5)
Record 3	B	DDD	7	(new observation)

Output A2.6 on page 125 shows the result with a BY key.

Output A2.6 Results with a BY Key

Data Fields	DF1	DF2		
		DF21	DF22	
Record 1	A	CCC	1	(obs 1)
		CCC	2	(obs 2)
		CCC	4	(obs 3)
Record 2	B	DDD	3	(obs 4)
		DDD	5	(obs 5)
		DDD	7	(new observation)

BY Key Considerations

When specifying BY keys for your view descriptors, keep in mind the following considerations:

- A duplicate consecutive observation results in an additional occurrence in any periodic group in the view descriptor.
- If you do an insert from an observation that has all missing values, the interface view engine inserts a record that is equivalent to all zeros and blanks.
- The APPEND function of the SAS Screen Control Language (SCL) must be preceded by a call to the SET function. Otherwise, APPEND inserts an observation that is equivalent to all zeros and blanks because the insert is too ambiguous for the interface view engine to resolve.
- If a view descriptor includes a periodic group and you try to add an observation that is another occurrence in that periodic group, the add may fail if you are attempting to add more occurrences than the periodic group field definition allows. One of the following will occur, depending on whether a BY key is specified:
  - If no BY key is defined, and

- □ if the last observation was not created from the periodic group, a new logical record is added.
- □ if the last observation was created from the periodic group, the add fails with a return code, and a new record is then added.

□ If a BY key is defined and the periodic group is selected to have an added occurrence, the add fails and a message displays.

Missing Values (Nulls)

When the interface view engine is reading ADABAS data and constructing an observation, it could find missing (null) values for data fields within an observation.

The interface view engine uses the L1, L2, L3, and L4 commands to retrieve ADABAS data. The values are returned in the record buffer using the standard length and format defined for that field. (Standard length is not used if you have specified a value for the DB Content field or the field is a variable length field.) If the field's value is null, the data are returned in the format in effect for that field.

Formats and their corresponding null values are listed below.

Format	Null Value
Alphanumeric	blanks
Binary	binary zeros
Fixed Point	binary zeros
Unpacked Decimal	unpacked decimal zeros
Packed Decimal	packed decimal zeros
Floating point	binary zeros

When an ADABAS record is read, the interface view engine is unable to tell whether a field has a value of zero (for numeric fields) or blanks (for alphanumeric fields) or truly has a null value. This is also true when you are updating. When you are using the FSEDIT procedure, if a value of zero or missing is used to modify an existing record, zeros are placed in the ADABAS record buffer and subsequently added to the ADABAS file. Blanks are placed in the record buffer if a blank or missing value was supplied for an alphanumeric field.

Since SAS System missing values are stored as zeros and blanks in ADABAS files, some SAS WHERE clauses are also impacted. For example, if either of the following SAS WHERE clauses are issued,

```
where aa is missing;
where aa is null;
```

the resulting condition is sent to ADABAS:

```
where aa = 0 (numeric)
where aa = '∅' (alphanumeric)
```

Note: Null values are processed differently by ADABAS if the ADABAS descriptor used in a WHERE clause has the Null Value Suppress (NU) definition option defined for it. △

Using Multiple-Value Fields in Selection Criteria

A multiple-value field can have 0 to 191 values per record, and ADABAS assigns an occurrence number to each value. When you include a multiple-value field in SAS/ACCESS descriptor files, you can use SAS variables that reference individual occurrences and a SAS variable that references all occurrences to perform special WHERE clause queries.

Table A2.1 on page 127 lists whether you can use a multiple-value field or its corresponding SAS variables in the SAS and view WHERE clauses.

Table A2.1 Multiple-Value Fields in WHERE Clauses

Multiple-Value Field	SAS WHERE Clause	View WHERE Clause
ADABAS data field name	no	yes
SAS name for individual Occurrence variable	yes	no
_ANY variable	yes	yes

WHERE Clause Examples

Using the multiple-value data field BRANCH-OFFICE from the CUSTOMERS DDM, the following examples illustrate using a multiple-value field in WHERE clauses.

Example 1

In a view WHERE clause, you can reference an ADABAS multiple-value field name, but you cannot do so in a SAS WHERE clause. For example, with the following WHERE clause in a view descriptor, the interface view engine searches all values of the multiple-value field:

```
where branch-office='LONDON'
```

The view WHERE clause produces the results in Output A2.7 on page 127.

Output A2.7 Output with ADABAS Multiple-Value Field Name in View WHERE Clause

OBS	CUSTNUM	BR_ANY	BRANCH_1	BRANCH_2	BRANCH_3	BRANCH_4
1	14324742		TORONTO	HOUSTON	TOKYO	LONDON
2	26422096		LONDON	NEW YORK		
3	26984578		LONDON	NEW YORK	ROME	
4	27654351		LONDON	BOSTON		
5	28710427		LONDON			

Example 2

You can use the individual occurrence SAS variables created by the ACCESS procedure such as BRANCH_1, BRANCH_2, and so on, in SAS WHERE clauses, but you cannot use them in a view WHERE clause. Note that individual occurrence conditions must be processed by the SAS System after ADABAS has completed its selection processing.

For example, the following SAS WHERE clause searches the second occurrence for BRANCH-OFFICE and retrieves the London values. The SAS System post-processes all records returned from the interface view engine to see if they meet the SAS WHERE clause in effect.

```
where branch_1='LONDON'
```

The SAS WHERE clause produces the results in Output A2.8 on page 128.

Output A2.8 Output with the Individual Occurrence SAS Variable in SAS WHERE Clause

OBS	CUSTNUM	BR_ANY	BRANCH_1	BRANCH_2	BRANCH_3	BRANCH_4
1	26422096		LONDON	NEW YORK		
2	26984578		LONDON	NEW YORK	ROME	
3	27654351		LONDON	BOSTON		
4	28710427		LONDON			

Example 3

You can use the _ANY variable created by the ACCESS procedure in both a SAS WHERE clause and a view WHERE clause. However, if you use the _ANY variable in a SAS WHERE clause, the ADABAS interface view engine must be able to process the entire SAS WHERE clause.

For example, with the following WHERE clause, the engine searches all occurrences of the multiple-value field:

```
where br_any = 'LONDON'
```

Whether that WHERE clause is a SAS WHERE clause or a view WHERE clause, the results in Output A2.9 on page 129 are produced. They are the same as for Output A2.7 on page 127.

Output A2.9 Output with _ANY Variable in View or SAS WHERE Clause

OBS	CUSTNUM	BR_ANY	BRANCH_1	BRANCH_2	BRANCH_3	BRANCH_4
1	14324742		TORONTO	HOUSTON	TOKYO	LONDON
2	26422096		LONDON	NEW YORK		
3	26984578		LONDON	NEW YORK	ROME	
4	27654351		LONDON	BOSTON		
5	28710427		LONDON			

This functionality prevents you from having to enter repetitive selection criteria such as

```
where branch_1='LONDON' or branch_2='LONDON'
      or branch_3='LONDON' ...
```

Using Periodic Group Fields in Selection Criteria

For an ADABAS periodic group data field, the ACCESS procedure automatically creates a SAS variable for the occurrence number within the periodic group. For example, the NATURAL DDM named CUSTOMERS has a periodic group field named SIGNATURE-LIST, which groups data fields LIMIT and SIGNATURE. The ACCESS procedure creates a SAS variable named SL_OCCUR for the occurrence numbers in LIMIT and SIGNATURE.

By including the _OCCUR variable in a view descriptor, you can retrieve the occurrence numbers for the periodic group. You can also include the _OCCUR variable in SAS WHERE clauses to qualify data, but the condition is processed by the SAS System after ADABAS has completed its selection processing. You cannot update the occurrence values, and you cannot use the _OCCUR variable in a view WHERE clause.

Table A2.2 on page 129 lists whether you can use periodic group SAS variable names, periodic group occurrence syntax, and a periodic group's corresponding _OCCUR variable in SAS and view WHERE clauses.

Table A2.2 Periodic Group Fields in WHERE Clauses

Periodic Group Field	SAS WHERE Clause	View WHERE Clause
SAS variable name	yes	yes
ADABAS data field name and occurrence syntax	no	yes
_OCCUR variable	yes	no

WHERE Clause Examples

Using the periodic group data field LIMIT from the CUSTOMERS DDM, the following examples illustrate using a periodic group data field in WHERE clauses.

Example 1

You can use the SAS variable name of a data field within a periodic group in both a SAS WHERE clause and a view WHERE clause. However, they will not always produce the same results because the SAS WHERE clause post-processes the results and, using the following example, looks at the value of variable LIMIT to determine whether it's equal to 5000. The view WHERE clause is not post-processed; when you use a periodic group field, ADABAS qualifies all periodic group occurrence values if any one meets the WHERE clause criteria.

For example, you can include the following WHERE clause in a view descriptor, and you can issue it as a SAS WHERE clause:

```
where limit = 5000
```

Stored in a view descriptor, the WHERE clause produces the results in Output A2.10 on page 130:

Output A2.10 Periodic Group Data Field Referenced in View WHERE Clause

OBS	CUSTNUM	SL_OCCUR	LIMIT
1	12345678	1	5000.00
2	14324742	1	5000.00
3	14324742	2	25000.00
4	14569877	1	5000.00
5	14569877	2	100000.00
6	19783482	1	5000.00
7	19783482	2	10000.00
8	26422096	1	5000.00
9	26422096	2	10000.00
10	27654351	1	5000.00
11	29834248	1	5000.00

However, as a SAS WHERE clause, the results in Output A2.11 on page 131 are produced.

Output A2.11 Periodic Group Data Field Referenced in SAS WHERE Clause

OBS	CUSTNUM	SL_OCCUR	LIMIT
1	12345678	1	5000.00
2	14324742	1	5000.00
3	14569877	1	5000.00
4	19783482	1	5000.00
5	26422096	1	5000.00
6	27654351	1	5000.00
7	29834248	1	5000.00
8	43459747	2	5000.00

Example 2

You can qualify a specific occurrence of a periodic group with a view WHERE clause, but only by using the periodic group occurrence syntax. However, all of the periodic group occurrence values for the qualified records are returned, not just the individual occurrence specified in the view WHERE clause. You cannot specify the occurrence syntax in a SAS WHERE clause. For example, this view WHERE clause produces the results in Output A2.12 on page 131.

```
where limit(2) = 5000
```

Output A2.12 Periodic Group Occurrence Syntax in View WHERE Clause

OBS	CUSTNUM	SL_OCCUR	LIMIT
1	43459747	1	1000.00
2	43459747	2	5000.00

Example 3

If you include the _OCCUR SAS variable in the view descriptor, you can use it in a SAS WHERE clause to specify an occurrence. However, you cannot use the _OCCUR variable in a view WHERE clause.

For example, this SAS WHERE clause produces the results shown in Output A2.13 on page 132.

```
where sl_occur = 2
```

Output A2.13 Periodic Group _OCCUR SAS Variable in SAS WHERE Clause

OBS	CUSTNUM	SL_OCCUR	LIMIT
1	14324742	2	25000.00
2	14569877	2	100000.00
3	14898029	2	50000.00
4	18543489	2	50000.00
5	19783482	2	10000.00
6	19876078	2	25000.00
7	26422096	2	10000.00
8	43459747	2	5000.00

To qualify the data even further, you could use this SAS WHERE clause, which produces the results in Output A2.14 on page 132.

```
where sl_occur = 2 and limit = 5000
```

Output A2.14 Periodic Group _OCCUR Variable and Occurrence Syntax in SAS WHERE Clause

OBS	CUSTNUM	SL_OCCUR	LIMIT
1	43459747	2	5000.00

Using a SAS WHERE Clause for Selection Criteria

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

When you specify a SAS WHERE clause, the SAS/ACCESS interface view engine translates those conditions into view WHERE clause conditions. Then, if the view descriptor includes a WHERE clause, the interface view engine connects the conditions with the Boolean operator AND. By default, the SAS WHERE clause conditions are connected before the view WHERE clause 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

```
(position=marketing) and (sex=female)
```

When the interface view engine translates SAS WHERE clause conditions into view WHERE clause conditions, some SAS WHERE clause capabilities are not available in a view WHERE clause. That is, some SAS WHERE clauses cannot be totally satisfied by the interface view engine.

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. In the example above, the engine 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 logical records as possible to improve performance.

Any conditions that are not supported are bypassed by the interface view engine, and post-processing (handled automatically by the SAS System) is required after the engine does its subsetting. The engine bypasses

- unacceptable conditions.
- conditions connected with OR to unacceptable conditions.

In Table A2.3 on page 133, assume DF1, DF2, and DF3 are ADABAS data fields referenced by a view descriptor. Remember that the SAS System never sees view WHERE clauses.

Table A2.3 Periodic Group Fields in WHERE Clauses

SAS WHERE Clause	View WHERE Clause	Translation	Processing Required?
DF2=B OR DF3>DF4+10	(DF1=A)	(DF1=A)	Yes
DF2=B & DF3>DF4+10	DF1=A	(DF2=B) & (DF1=A)	Yes
DF2=B & DF3>C	DF1=A	(DF2=B) & (DF3>C) & (DF1=A)	No
DF2=B OR DF3>C	DF1=A	(DF2=B) OR (DF3>C) & (DF1=A)	No

SAS WHERE Clause Conditions Acceptable to ADABAS

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

- The operators are translated as shown in Table A2.4 on page 134.

Table A2.4 Translating acceptable SAS WHERE clause conditions into view WHERE clause conditions

SAS WHERE Clause Syntax	View WHERE Clause Translation
=	=
>	>
<	<
<>	!=
≥	≥
≤	≤
(	(
)	)
AND	AND
OR	OR

- The interface view engine also translates BETWEEN and IN conditions and the date format (if a SAS format is supplied in the DB Content field).

Table A2.5 Translating BETWEEN and IN condtions and the date format

SAS WHERE Clause Syntax	View WHERE Clause Translation
DF1 BETWEEN 1 AND 3	(DF1 >= 1 AND DF1 <= 3)
DF1 IN (4,9,14)	DF1=4 OR DF1=9 or DF1=14
DF4 = '02AUG87'D	DF4 = 870802

SAS WHERE Clause Conditions Not Acceptable to ADABAS

Any SAS WHERE clause conditions that are not acceptable to the ADABAS interface view engine are handled automatically by SAS post-processing. Following are some (but not all) of those conditions:

- item-to-item comparison
- pattern matching
- arithmetic expressions, for example,

```
WHERE DF1 = DF4 * 3
WHERE DF4 < - DF5
```

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

```
WHERE DF1
WHERE (DF1 = DF2) * 20
```

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

```
DF1 =: ABC
```

- DATETIME and TIME formats, for example,

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

- SOUNDEX
- HAVING, GROUP BY
- NOT CONTAINS.

When a SAS WHERE Clause Must Reference Descriptor Data Fields

When you are using a SAS WHERE clause, a referenced ADABAS data field must be an ADABAS descriptor in the following situations:

- The SAS WHERE clause contains more than one condition.
- The SAS WHERE clause uses the SPANS or NE operator.
- You are also planning to issue a SAS BY statement or a SAS ORDER BY clause.
- The view descriptor also includes a view SORT clause.
- The view descriptor also includes a view WHERE clause.

Deciding How to Specify Selection Criteria

Use the following guidelines to determine when to use a SAS WHERE clause and when to use a view WHERE clause.

View WHERE Clause

Include a WHERE clause in your view descriptor when you want to

- restrict users of view descriptors to certain subsets of data
- prevent users from sequentially passing all the ADABAS data
- use syntax not available in the SAS WHERE clause, such as periodic group occurrence syntax or multiple-value compares.

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 the view WHERE clause does not support, such as arithmetic expressions, truncated comparisons, or pattern matching
- use conditions on fabricated data fields such as ISN, periodic group _OCCUR variables, or any individually selected multiple-value fields
- combine AND/OR conditions using non-descriptor data fields.

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

SAS/ACCESS® Interface to ADABAS Software: Reference, Version 8

Copyright © 1999 by SAS Institute Inc., Cary, NC, USA.

ISBN 1–58025–546–9

All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, by any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute, Inc.

U.S. Government Restricted Rights Notice. Use, duplication, or disclosure of the software by the government is subject to restrictions as set forth in FAR 52.227–19 Commercial Computer Software-Restricted Rights (June 1987).

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

1st printing, October 1999

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

The Institute is a private company devoted to the support and further development of its software and related services.