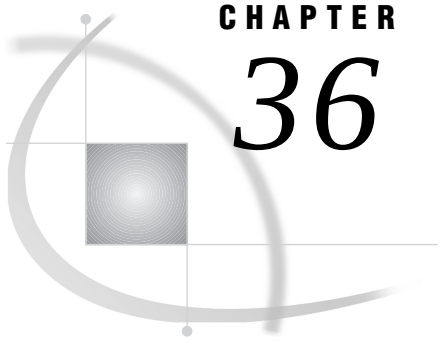

CHAPTER

36

SAS Component Language (SCL) Interface to Message Attachments

Introduction

Both direct and indirect messaging services allow attachments to be included with messages that are being sent between a client and a server portion of an application. When a query surfaces a message event, it also surfaces an attachment list if an attachment list was included with the message. Only the attachment list is surfaced by the query; no attachments have actually been transferred at this point. This chapter explains how to *send* and *receive* attachments in the SCL environment, as well as how to handle *error conditions* that are associated with message attachments.

Sending Attachments

To include attachments with the sending of a message, an attachment list must be built to indicate what attachments to include. Both the _send and _sendlist methods (for the Cnction and the Queue classes) use an *attachlist* parameter.

The *attachlist* parameter is an SCL list that contains other SCL lists and must contain a separate list for each attachment to include with the message. The syntax for building the attachment list is the same, whether using direct or indirect (queued) messaging.

The supported attachment types are:

- Data Set
- External File
- Catalog
- MDDB
- DMDB
- FDB
- SQL Views

Data Set Attachments

When data set attachments are transferred, all data set attributes are cloned by default. These include label, type, passwords, encryption, index, and sort order information.

To specify a data set attachment, the following named items are *required*:

TYPE
: The value of this named item must be "DATASET" to indicate that this is a data set attachment.

MEMNAME
: The value of this named item should be the data set's member name.

LIBNAME
: The value of this named item should be the data set's library name.

Additionally, there are optional named items that may be specified for data set attachments. These optional settings not only provide a means to subset the data before transfer, but they also provide a way to surface descriptive information to the receiving side. The *optional* named items that are supported include:

DESCRIPTION
: The value of this named item is user-specified, descriptive text to accompany the attachment. This information is surfaced to the user on the receiving side and allows the user to provide specific information about this attachment.

WHERE
: The value of this named item is a WHERE statement to apply to the data set. This provides a way to subset the data before transfer.

DATASET_OPTIONS
: The value of this named item can be any valid data set options string. This too provides a way to subset the data before transfer.

ATTACH_VERSION
: The value of this named item is a character string and the only supported value in Version 7 or Version 8 is VERSION_612. This parameter only applies to indirect

messaging and only applies when sending catalog or data set attachments. If a Version 7 or Version 8 application sends a data set or a catalog attachment to a queue, a Version 6 application cannot accept them; it will fail. However, the Version 7 or Version 8 application can specify this option to indicate that the catalog or data set should be sent to the queue in Version 6 format. This way, both Version 6 and Version 7 or Version 8 applications can accept the attachment.

MAJOR_VERSION
The value of this named item is a numeric, user-specified version that will be presented to the receiver.

MINOR_VERSION
The value of this named item is a numeric, user-specified version that will be presented to the receiver.

INDEX
The value of this named item must be either N or NO. By default, the data set's index is re-created on the output data set. This named item allows the default to be overridden so that indexes are not created on the output file.

Example 1

This example specifies two data set attachments to be included with the message. The data sets are SASUSER.DATAX and WORK.ABC.

For data set SASUSER.DATAX, the required named items would be defined as:

- named item TYPE has a value of DATASET
- named item LIBNAME has a value of SASUSER
- named item MEMNAME has a value of DATAX.

For data set WORK.ABC, the required named items would be defined as:

- named item TYPE has a value of DATASET
- named item LIBNAME has a value of WORK
- named item MEMNAME has a value of ABC.

For both data sets, DESCRIPTION was *optionally* set so that more descriptive information will be surfaced to the receiver.

```
   /*************************************/
   /* list1 contains attachment one,    */
   /* sasuser.datax                     */
   /*************************************/
  list1 = makelist();
  rc = setnitemc(list1, "DATASET", "TYPE");
  rc = setnitemc(list1, "DATAX", "MEMNAME");
  rc = setnitemc(list1, "SASUSER", "LIBNAME");
  rc = setnitemc(list1, "Tasklist dataset for
                client application.",
                "DESCRIPTION");

   /*************************************/
   /* list2 contains attachment two,    */
   /* work.abc                          */
   /*************************************/
  list2 = makelist();
  rc = setnitemc(list2, "ABC", "MEMNAME");
  rc = setnitemc(list2, "WORK", "LIBNAME");
```

```
rc = setnitemc(list2, "DATASET", "TYPE");
rc = setnitemc(list2, "Playpen data.",
               "DESCRIPTION");

   /*************************************/
   /* attachlist is the main attachment   */
   /* list. It must contain a separate    */
   /* SCL list for each attachment        */
   /*************************************/
attachlist = makelist();
attachlist = insertl(attachlist, list1, -1);
attachlist = insertl(attachlist, list2, -1);

msgtype = 25;
hdr = 0;
call send(obj, '_sendlist', msgtype, hdr,
          attachlist, rc, sendlist);

             /***  or  ***/

call send(obj, '_send', msgtype, hdr,
          attachlist, rc, string1);
```

Example 2

This example illustrates the use of WHERE and DATASET_OPTIONS to subset the data before transfer.

```
   /*************************************/
   /* list 1 contains attachment one,     */
   /* sasuser.xx                          */
   /*************************************/
list1 = makelist();
rc = setnitemc(list1, "DATASET", "TYPE");
rc = setnitemc(list1, "XX", "MEMNAME");
rc = setnitemc(list1, "SASUSER", "LIBNAME");
rc = setnitemc(list1, "(X>10) AND (Y < 100)",
               "WHERE");
rc = setnitemc(list1, "DROP=NAMES READ=X",
               "DATASET_OPTIONS");

attachlist = makelist();
attachlist = insertl(attachlist, list1, -1);

msgtype  = 2;
hdr = 0;

call send(cobs, '_send', msgtype, hdr,
          attachlist, rc, Numeric1);
```

Catalog Attachments

SAS catalogs are another type of attachment that may be included with messages. To specify a catalog attachment, the following named items are *required*:

TYPE
The value of this named item must be CATALOG to indicate that this is a catalog attachment.

MEMNAME
The value of this named item should be the catalog's member name.

LIBNAME
The value of this named item should be the catalog's library name.

Additionally, there are optional named items that may be specified for catalog attachments. These optional settings provide a way to subset the catalog before transfer and to surface descriptive information to the receiving side. The *optional* named items that are supported include:

DESCRIPTION
The value of this named item is user-specified, descriptive text to accompany the attachment. This information is surfaced to the user on the receiving side and allows the user to provide specific information about this attachment.

SELECT
The value of this named item is a SELECT statement to apply to the catalog. This provides a way to select specific entries to include without sending the entire catalog. The value of this named item should take the form of ENTRY1.ENTRYTYPE ENTRY2.ENTRYTYPE..., for each entry to select. The select and exclude items are mutually exclusive so you cannot specify both SELECT and EXCLUDE for a given catalog attachment.

EXCLUDE
The value of this named item is an EXCLUDE statement to apply to the catalog. This provides a way to exclude specific entries so that unnecessary entries are not transferred. The value of this named item should take the form of ENTRY1.ENTRYTYPE ENTRY2.ENTRYTYPE..., for each entry to exclude. The select and exclude items are mutually exclusive so you cannot specify both SELECT and EXCLUDE for a given catalog attachment.

MAJOR_VERSION
The value of this named item is a numeric, user-specified version that will be presented to the receiver.

MINOR_VERSION
The value of this named item is a numeric, user-specified version that will be presented to the receiver.

Example

This example specifies one catalog attachment to be included with the message. The catalog is SASHELP.BASE. The *required* named items are

- named item TYPE has a value of CATALOG
- named item LIBNAME has a value of SASHELP
- named item MEMNAME has a value of BASE.

Optionally, DESCRIPTION is specified to provide descriptive information to the receiving side, and SELECT is specified to select specific entries to include.

```
   /*************************************/
   /* main attachment list              */
   /*************************************/
attachlist = makelist();
```

```
   /************************************/
   /* build attachment one with the       */
   /* SELECT statement                    */
   /************************************/
list1 = makelist();
rc = setnitemc(list1, "CATALOG", "TYPE");
rc = setnitemc(list1, "BASE", "MEMNAME");
rc = setnitemc(list1, "SASHELP", "LIBNAME");
rc = setnitemc(list1, "RLIST.LIST XYZ.SCL
               MAIN.FRAME",
               "SELECT");
rc = setnitemc(list1, "A few entries
               from base catalog.",
               "DESCRIPTION");

   /************************************/
   /* insert attachment one into main list*/
   /************************************/
attachlist = insertl(attachlist, list1, -1);

msgtype = 2;
hdr = 0;
call send(cobs, '_send', msgtype, hdr,
          attachlist, rc, "string1");
```

External File Attachments

External files may also be sent as attachments. To specify an external file attachment, the following named items are *required*:

TYPE
: The value of this named item must be either EXTERNAL_TEXT or EXTERNAL_BIN. If the attachment is a text file, specify EXTERNAL_TEXT so that the appropriate translation will occur. If including a binary file as an attachment, specify EXTERNAL_BIN.

FILENAME or FILEREF
: If the external file is to be referenced by a fileref, specify the named item FILEREF; its value will be the fileref that defines this file. If the external file is to be referenced by its physical filename, specify the named item FILENAME; its value will be the physical name of the file.

Additionally, there are optional named items that may be specified. These *optional* settings include:

DESCRIPTION
: The value of this named item is user-specified, descriptive text to accompany the attachment. This information is surfaced to the user on the receiving side and allows the user to provide specific information about this attachment.

MAJOR_VERSION
: The value of this named item is a numeric, user-specified version that will be presented to the receiver.

MINOR_VERSION
: The value of this named item is a numeric, user-specified version that will be presented to the receiver.

Example

This example defines three external file attachments to send with the message. The first attachment is an external text file, **/tmp/text.file**. The required named items for this attachment are

- named item TYPE has a value of EXTERNAL_TEXT
- named item FILENAME has a value of **/tmp/text.file**

The second attachment is an external text file defined by the fileref RLINK. The required named items for this attachment are

- named item TYPE has a value of EXTERNAL_TEXT
- named item FILEREF has a value of RLINK.

Finally, the last attachment is an external binary file, **/tmp/binary.file**. The required named items for this attachment are

- named item TYPE has a value of EXTERNAL_BIN
- named item FILENAME has a value of **/tmp/binary.file**.

The optional named items that may be specified for external file attachments, which are used in this example, include DESCRIPTION and MAJOR_VERSION.

```
   /*************************************/
   /* LIST1 contains attachment one     */
   /*************************************/
list1 = makelist();
rc = setnitemc(list1, "EXTERNAL_TEXT", "TYPE");
rc = setnitemc(list1, "/tmp/text.file", "FILENAME");
rc = setnitemc(list1, "modified script file for unix",
            "DESCRIPTION");
rc = setnitemc(list1, 11, "MAJOR_VERSION");

   /*************************************/
   /* LIST2 contains attachment two     */
   /*************************************/
list2 = makelist();
rc = setnitemc(list2, "EXTERNAL_TEXT", "TYPE");
rc = setnitemc(list2, "RLINK", "FILEREF");

   /*************************************/
   /* LIST3 contains attachment three   */
   /*************************************/
list3 = makelist();
rc = setnitemc(list3, "EXTERNAL_BIN", "TYPE");
rc = setnitemc(list3, "/tmp/binary.file", "FILENAME");

   /*************************************/
   /* ATTACHLIST is the main attachment */
   /* list; insert each attachment into */
   /* main attachment list.             */
   /*************************************/
attachlist = makelist();
attachlist = insertl(attachlist, list1, -1);
attachlist = insertl(attachlist, list2, -1);
attachlist = insertl(attachlist, list3, -1);

msgtype = 22;
```

```
hdr = 0;
call send(cobs, '_send', msgtype, hdr,
          attachlist, rc, string1);
```

Utility Attachments

Utility files that are supported include files of the following types: MDDB, DMDB, FDB, and SQL Views.

To specify a utility file attachment, the following named items are *required*:

TYPE
The value of this named item must be either MDDB, DMDB, VIEW, or FDB to indicate the type of attachment.

MEMNAME
The value of this named item should be the member name.

LIBNAME
The value of this named item should be the library name.

Additionally, there are optional named items that may be specified for utility file attachments. These optional settings provide a way to surface descriptive information to the receiving side. The *optional* named items that are supported include:

DESCRIPTION
The value of this named item is user-specified, descriptive text to accompany the attachment. This information is surfaced to the user on the receiving side and allows the user to provide specific information about this attachment.

MAJOR_VERSION
The value of this named item is a numeric, user-specified version that will be presented to the receiver.

MINOR_VERSION
The value of this named item is a numeric, user-specified version that will be presented to the receiver.

Example

This example specifies two utility file attachments and one data set to be included with the message. The files are SASUSER.TESTDATA (a SAS data set), WORK.FMDDB (an MDDB file), and SASUSER.TESTVIEW (an SQL View).

The required named items for the data set SASUSER.TESTDATA are

- named item TYPE has a value of DATASET
- named item LIBNAME has a value of SASUSER
- named item MEMNAME has a value of TESTDATA.

The required named items for the data set WORK.FMDDB are

- named item TYPE has a value of MDDB
- named item LIBNAME has a value of WORK
- named item MEMNAME has a value of FMDDB.

The required named items for the data set SASUSER.TESTVIEW are

- named item TYPE has a value of VIEW
- named item LIBNAME has a value of SASUSER
- named item MEMNAME has a value of TESTVIEW.

For the three data sets, DESCRIPTION was optionally set so that more descriptive information will be surfaced to the receiver.

```
list1 = makelist();
rc = setnitemc(list1, "DATASET", "TYPE");
rc = setnitemc(list1, "TESTDATA","MEMNAME");
rc = setnitemc(list1, "SASUSER", "LIBNAME");
rc = setnitemc(list1,
           "Testdata in sasuser directory.",
           "DESCRIPTION");

list2 = makelist();
rc = setnitemc(list2, "FMDDB", "MEMNAME");
rc = setnitemc(list2, "WORK", "LIBNAME");
rc = setnitemc(list2, "MDDB", "TYPE");
rc = setnitemc(list2, "Playpen mddb",
              "DESCRIPTION");

list3 = makelist();
rc = setnitemc(list3, "TESTVIEW","MEMNAME");
rc = setnitemc(list3, "SASUSER", "LIBNAME");
rc = setnitemc(list3, "VIEW", "TYPE");
rc = setnitemc(list3, "SQL view file",
              "DESCRIPTION");

   /*************************************/
   /* ATTACHLIST is the main attachment   */
   /* list. It must contain a separate    */
   /* SCL list for each attachment        */
   /*************************************/
attachlist = makelist();
attachlist = insertl(attachlist, list1, -1);
attachlist = insertl(attachlist, list2, -1);
attachlist = insertl(attachlist, list3, -1);

msgtype = 25;
hdr = 0;
call send(obj, '_sendlist', msgtype, hdr,
          attachlist rc, sendlist);

                /***  or  ***/

call send(obj, '_send', msgtype, hdr,
          attachlist rc, string1);
```

Accepting Attachments

When a _query surfaces a message that includes attachments, the *attachlist* parameter will be non-empty, and it will mirror the attachment list specified on the sending side, with a few changes. First, the receiving application is not made aware of any options that may have been used to subset the data (for example, KEEP, EXCLUDE, WHERE, and data set options). Also, each attachment will have the

additional named item ATTACH_ID that is used to identify which attachments are to be accepted or received.

When the _query surfaces the message and its attachment list, *the attachments have not yet been transferred*.

Note: The receiver is responsible for deciding which attachments to receive by invoking the _acceptAttachment method that has a valid value for the *attachlist* parameter. This method initiates the transfer of the specified attachments. If no attachments are to be accepted, set *attachlist* to 0 when calling the _acceptAttachment method. △

The accept method supports the COMPLETE flag, which indicates that attachment acceptance is complete. This is an optional flag that does not have to be set on every call to _acceptAttachment. However, whenever a query surfaces a non-empty attachment list, _acceptAttachment with the COMPLETE flag must be called at some point to signal the completion of attachment receipt. No subsequent queries and/or sends will be allowed until the COMPLETE flag is set on the accept method.

When building the attachment list on the sending side, the main attachment list parameter contained a separate list for each attachment. The same is required by _acceptAttachment. The *attachlist* parameter is required to be an SCL list that contains other SCL lists; one for each attachment to accept.

Data Set Attachments

A list to accept a SAS data set must include the following *required* named items:

OUTLIB
: The value of this named item should be the output library name.

OUT
: The value of this named item should be the output member name.

ATTACH_ID
: The value of this named item should be a numeric identifier that indicates which attachment to receive. This is the attachment id that is surfaced by the query in the *attachlist* parameter.

Example

When the query returns with a message and its attachment list, the *attachlist* parameter contains three attachments.

List one, first item in *attachlist*:

- named item TYPE has a value of DATASET
- named item MEMNAME has a value of NAMES
- named item LIBNAME has a value of SASUSER
- named item ATTACH_ID has a value of 1.

List two, second item in *attachlist*:

- named item TYPE has a value of CATALOG
- named item MEMNAME has a value of CONNECT
- named item LIBNAME has a value of SASHELP
- named item ATTACH_ID has a value of 2.

List three, third item in *attachlist*:

- named item TYPE has a value of DATASET

- named item MEMNAME has a value of EMPLOYEES
- named item LIBNAME has a value of WORK
- named item ATTACH_ID has a value of 3.

This example accepts two of the three attachments by making two separate calls to the accept method. First, the contents of the third attachment in *attachlist* (which is actually the first item) is received by specifying ATTACH_ID equal to 3. This transfers the input data set WORK.EMPLOYEES into the output data set WORK.ABC. The second attachment to be received is identified by specifying ATTACH_ID equal to 1. This transfers the input data set SASUSER.NAMES into the output data set SASUSER.NAMES.

```
alist = makelist();
att1  = makelist();
rc = setnitemn(att1, 3, "ATTACH_ID");
rc = setnitemn(att1, "WORK", "OUTLIB");
rc = setnitemn(att1,"ABC", "OUT");
alist = insertl(alist, att1, -1);

   /*************************************/
   /* Accept this attachment but do not   */
   /* set the COMPLETE flag as there are  */
   /* additional ones to accept.          */
   /*************************************/
call send(Obj, '_acceptAttachment',
          alist, rc);

rc = dellist(alist, 'Y');
alist = makelist();
att2 = makelist();
rc = setnitemn(att2, 1, "ATTACH_ID");
rc = setnitemn(att2, "SASUSER", "OUTLIB");
rc = setnitemn(att2,"NAMES", "OUT");
alist = insertl(alist, att2, -1);

   /*************************************/
   /* The COMPLETE flag is set to         */
   /* indicate attachment acceptance is   */
   /* complete after the second attachment */
   /* is transferred.                     */
   /*************************************/
call send(Obj, '_acceptAttachment', alist,
          rc, "COMPLETE");
```

Catalog Attachments

To receive catalog attachments, the following named items are *required*:

OUTLIB
The value of this named item should be the output library name.

OUT
The value of this named item should be the output member name.

ATTACH_ID
The value of this named item should be a numeric identifier that indicates which attachment to receive. This is the attachment id that is surfaced by the query in the *attachlist* parameter.

Example

When the query returns with a message and its attachment list, the *attachlist* parameter contains the following three attachments:

List one, first item in *attachlist*:

- named item TYPE has a value of CATALOG
- named item MEMNAME has a value of BASE
- named item LIBNAME has a value of SASHELP
- named item ATTACH_ID has a value of 1.

List two, second item in *attachlist*:

- named item TYPE has a value of DATASET
- named item MEMNAME has a value of WORK
- named item LIBNAME has a value of TEMP
- named item ATTACH_ID has a value of 2.

List three, third item in *attachlist*:

- named item TYPE has a value of CATALOG
- named item MEMNAME has a value of INFOCAT
- named item LIBNAME has a value of WORK
- named item ATTACH_ID has a value of 3.

This example accepts two of the three attachments by making one call to the accept method. The first attachment to receive is identified by setting ATTACH_ID equal to 3. This transfers catalog WORK.INFOCAT to the output catalog, SASUSER.INFOCAT. The second attachment to receive is identified by setting ATTACH_ID equal to 1. This transfers catalog SASHELP.BASE to the output catalog LOCAL.TASKC.

```
alist = makelist();
att1 = makelist();
rc = setnitemn(att1, 3, "ATTACH_ID");
rc = setnitemn(att1, "SASUSER", "OUTLIB");
rc = setnitemn(att1,"INFOCAT", "OUT");

att2 = makelist();
rc = setnitemn(att2, 1, "ATTACH_ID");
rc = setnitemn(att2, "LOCAL", "OUTLIB");
rc = setnitemn(att2,"TASKC", "OUT");

   /*************************************/
   /* Insert the attachment lists into  */
   /* the main attachment list, alist.  */
   /*************************************/
alist = insertl(alist, att1, -1);
alist = insertl(alist, att2, -1);

   /*************************************/
   /* The COMPLETE flag is set to       */
   /* indicate attachment acceptance is */
```

```
   /* complete.                              */
   /*****************************************/
call send(Obj, '_acceptAttachment',
          alist, rc, "COMPLETE");
```

External File Attachments

To accept external file attachments, the following named items are *required*:

OUTFILE or OUTFILEREF
: The value of the named item OUTFILE should be the physical filename of the output file. The value of the named item OUTFILEREF should be the output fileref. Only one of these should be specified.

ATTACH_ID
: The value of this named item should be a numeric identifier that indicates which attachment to receive. This is the attachment id that is surfaced by the query in the *attachlist* parameter.

Example

When the query returns with a message and its attachment list, the *attachlist* parameter contains the following four attachments:

List one, first item in *attachlist*:

- named item TYPE has a value of DATASET
- named item MEMNAME has a value of NAMES
- named item LIBNAME has a value of SASUSER
- named item ATTACH_ID has a value of 1.

List two, second item in *attachlist*:

- named item TYPE has a value of EXTERNAL_TEXT
- named item MEMNAME has a value of **/tmp/notes.txt**
- named item ATTACH_ID has a value of 2.

List three, third item in *attachlist*:

- named item TYPE has a value of EXTERNAL_BIN
- named item MEMNAME has a value of BINREF
- named item ATTACH_ID has a value of 3.

List four, fourth item in *attachlist*:

- named item TYPE has a value of EXTERNAL_TEXT
- named item MEMNAME has a value of RLINK
- named item ATTACH_ID has a value of 4.

This example accepts three of the four attachments by making one call to the accept method. The first attachment to be received is identified by specifying ATTACH_ID equal to 2. This transfers the input file **/tmp/notes.txt** to the output file defined by the fileref AFILE. The second attachment to be received is identified by specifying ATTACH_ID equal to 3. This transfers the input file defined by the fileref BINREF, to the output file defined by the fileref BFILE. The third attachment to be received is identified by specifying ATTACH_ID equal to 4. This transfers the input file defined by the fileref RLINK, to the output file **/tmp/rlink.scr** data set SASUSER.NAMES.

```
alist = makelist();
att1 = makelist();
```

```
rc = setnitemn(att1, 2, "ATTACH_ID");
rc = setnitemn(att1, "afile", "OUTFILEREF");
alist = insertl(alist, att1, -1);

att2 = makelist();
rc = setnitemn(att2, 3, "ATTACH_ID");
rc = setnitemn(att2, "bfile", "OUTFILEREF");
alist = insertl(alist, att2, -1);

att3 = makelist();
rc = setnitemn(att3, 4, "ATTACH_ID");
rc = setnitemn(att3, "/tmp/rlink.scr",
               "OUTFILE");
alist = insertl(alist, att3, -1);

   /*************************************/
   /* accept attachments                */
   /*************************************/
call send(Obj, '_acceptAttachment', alist,
          rc);


   /*************************************/
   /* acceptance complete               */
   /*************************************/
call send(Obj, '_acceptAttachment', 0,
          rc, "COMPLETE");
```

Attachment Error Handling

Transfer Errors: Direct vs Indirect

When sending a message to a message queue, all attachments, along with the message, are transferred to the queue when the _send or _sendlist is invoked. The attachments are stored at the DOMAIN server until fetched by a user. If an error occurs while sending the attachments to the queue, neither the message nor the attachments will be delivered to the queue. In this scenario, the return code from _send or _sendlist will be set to _SEATTXF. This is an error indicating that neither the message nor the attachments were delivered because one or more errors occurred during attachment transfer.

When a message is sent by using direct messaging, only the attachment list, along with the message, is sent to the receiving side initially. The receiver is then responsible for determining, which (if any), attachments should actually be transferred. Because the message is delivered to the receiver before any attachments are actually transferred, an error that is encountered during the attachment transfer will not cause the _send to terminate.

If an error is encountered, the current attachment transfer is terminated, but the remaining attachments selected to be received are sent to the receiving side. If any errors are encountered during attachment transfer, the return code from _send or _sendlist will be set to _SWATTXF. This is only a warning that indicates which message was successfully sent, but one or more errors occurred during attachment transfer.

Accept Errors

When a message includes attachments, the receiver has the responsibility to determine (by using the _acceptAttachment method) which attachments are ultimately transferred. If an error is encountered during attachment transfer, the current attachment transfer is terminated, but the transfer continues with the next attachment in *attachlist*. If any errors are encountered, the return code from _acceptAttachment is set to _SWATTXF. This is only a warning, which indicates that one or more errors occurred during attachment transfer. This behavior is the same for both direct and indirect messaging.

Attachment Error Codes

To review what was mentioned above, a specific return code will be set if an error is encountered during attachment transfer.

- when sending on a Cnction instance, _SWATTXF is returned
- when sending on a Queue instance, _SEATTXF is returned
- when accepting attachments on either a Queue or a Cnction instance, _SWATTXF is returned

When one of these scenarios occurs, the *attachlist* parameter passed to these methods will be updated. An additional named item, *rc*, will be added to each separate attachment list. The value of *rc* will be a numeric return code that can be used to determine what caused the error for this particular attachment transfer. The defined return codes (*rc*) include:

Input File Errors (error occurred on input file):

20	general I/O error
21	libname does not exist
22	memname does not exist
23	invalid or missing password
24	invalid data set option value
25	invalid data set option name
26	general error parsing data set options
27	error parsing WHERE statement
28	bad physical filename
29	file in use
30	file does not exist
31	invalid authorization for external file
32	general open error
33	error retrieving integrity constraints
34	variable name contains unsupported characters or is too long
35	key name contains unsupported characters or is too long.

Output File Errors (error occurred on output file):

80	general I/O error

81	LIBNAME does not exist
82	invalid or missing password
83	bad physical filename
84	file in use
85	file does not exist
86	invalid authorization for external file
87	general open error
88	file already exists
89	integrity constraint creation failure
90	integrity constraint error.

General and Miscellaneous Errors:

1	out-of-memory error
2	open of catalog by queue manager failed
3	catalog read error encountered by queue manager
4	catalog write error encountered by queue manager
5	index create failure
6	backwards compatibility failure
7	unsupported view; only SQL views supported.

Example

In the following example, one attachment is accepted into a non-existent library name:

```
   /*************************************/
   /* Build one attachment list, att1.    */
   /*************************************/
att1 = makelist();
rc = setnitemc(att1, 1, "ATTACH_ID");
rc = setnitemc(att1, "NOEXIST", "OUTLIB");
rc = setnitemc(att1, "A", "OUT");

alist = makelist();
alist = insertl(alist, att1, -1);

call send(obj, "_acceptAttachment",
          alist, rc);

   /*************************************/
   /* If error, dump out attachment list  */
   /* to view rc.                         */
   /*************************************/
if (rc NE 0) then
   call putlist(alist, "Attachment list
                after accept:", 1);
```

After the accept method call, the attachment list *alist* has the following named items:

- named item ATTACH_ID has a value of 1
- named item OUTLIB has a value of NOEXIST
- named item OUT has a value of A
- named item RC has a value of 81.

Due to an error, the attachment has an additional named item (*rc*) that is set to 81 to indicate that the output library does not exist. Similarly, when the sender returns from the _send or _sendlist, the *attachlist* parameter will be updated with the *rc* named item to reflect that the attachment transfer failed.

```
att1 = makelist();
rc = setnitemc(att1, "SASUSER","LIBNAME");
rc = setnitemc(att1, "NAMES",  "MEMNAME");
rc = setnitemc(att1, "DATASET","TYPE");

attachlist = makelist();
attachlist = insertl(attachlist, att1, -1);

call send(cnctionObj, "_send", msgtype,
          attachlist, rc, "Message One");

if (%sysrc(_SWATTXF) = rc) then do;
   call putlist(attachlist,
                "attachlist after send", -1);
end;
```

Assuming that the attachment was accepted by the receiving side (as shown in the example) the attachment list (*attachlist*) after the send is updated with the *rc* named item to reflect that the attachment transfer failed.

- named item LIBNAME has a value of SASUSER
- named item MEMNAME has a value of NAMES
- named item TYPE has a value of DATASET
- named item RC has a value of 81.

Again, the attachment contains an additional named item (*rc* set to 81) to indicate any errors that occurred during the transfer.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/CONNECT User's Guide, Version 8*, Cary, NC: SAS Institute Inc., 1999. pp. 537.

SAS/CONNECT User's Guide, Version 8

ISBN 1–58025–477–2

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

1st printing, September 1999