



## CHAPTER

## 30

## Examples of Direct Messaging

---

<i>Introduction</i>	277
<i>Loading a Station Instance</i>	277
<i>Open a Station Instance</i>	277
<i>Query a Station</i>	278
<i>Query a Connection Object</i>	278
<i>Loading a Connection Instance</i>	278
<i>Open a Connection Instance</i>	278
<i>Send a Message</i>	279
<i>Receive a Message</i>	279
<i>Disconnect from a Server</i>	279
<i>Close Station Instance</i>	279

---

### Introduction

This chapter outlines the general steps for implementing a sample distributed application by using direct messaging.

---

### Loading a Station Instance

The first thing you must do in both the client and server portions of the SCL application is to set up a *station instance*.

```
stationid = loadclass('sashelp.connect.station');
stationInst = instance(stationid);
```

---

### Open a Station Instance

After you have your station instance, both the client and server portions of the SCL application must initialize the messaging environment by using the `_open` method to open the station instance.

```
call send(stationInst, '_open', 'payroll', rc);
```

*Note:* The client is only able to talk with its targeted server if their instance names match. In this example, the matching instance name is the third parameter, PAYROLL.

△

---

## Query a Station

After the station has been successfully opened, the server application can use the `_query` method to listen for incoming messages. A server application can either listen for messages from *any* user by doing a `_query` on a station, or a server application can listen for messages from a *specific* user by doing a `_query` on a connection object.

```
call send(stationInst, '_query', eventtype,
          msgtype, attach_list,
          connection_object, rc);
```

---

## Query a Connection Object

The event types returned by the `_query` method are CONNECT, MESSAGE, DISCONNECT, and ABORT.

If a CONNECT event is returned from the `_query` method, it means that a client application has initiated a connect so that it can communicate with the server application. The server application can then use the returned *connection\_object* parameter for subsequent queries from that specific user.

```
call send(connection_object, '_query', eventtype,
          msgtype, attach_list, rc);
```

If a MESSAGE event is returned by the `_query` method, the server receives the MESSAGE and responds.

If a DISCONNECT event is returned by the `_query` method, the specific client application has terminated the connection and the connection object is deleted.

---

## Loading a Connection Instance

In order for the client application to establish communication with the server application, it must set up a *connection instance*.

```
cnctid=loadclass('sashelp.connect.cnction');
connection=instance(cnctid);
```

---

## Open a Connection Instance

After you have your *connection instance*, the client portion of the SCL application must establish communication with the server application by using the `_open` method. The client must know the server's nickname in order to connect to it.

```
call send(connection, '_open',
          stationInst, '/shr1', rc);
```

*Note:* The station instance names in both the client and the server applications must match or the connection open will fail. In this example, the station instance name is the third parameter.  $\Delta$

---

## Send a Message

When the connection is successfully opened, the client application is ready to send messages by means of the `_send` method. Using the `_send` method, the client application can optionally specify a message type. The message-type values are defined by the application, and both the client and server pieces of the application must be coded to recognize and process this defined set of message types. For example, the message type can be used to indicate the number and the types of parameters in the message:

- Message type 1 indicates one parameter of type character (that is, a string).
- Message type 2 indicates one parameter of type numeric.

```
msgtype=1;
call send(payload, '_send', msgtype, rc,
          'Start nightly processing');
```

---

## Receive a Message

When the server detects that it has been sent a message, it attempts to receive the message. The message type may be used by the server to determine the type and number of parameters to fetch.

```
call send(connection_object, '_recv', rc, task_string);
```

---

## Disconnect from a Server

The `_query`, `_send`, and `_recv` methods can continue repeatedly until the particular needs of the application are accomplished. When the client is finished, it indicates this to the server by sending a `_disconnect`.

```
call send(connection, '_disconnect', rc);
```

---

## Close Station Instance

Both the client and the server shut down the messaging environment by executing the `_close` method.

```
call send(stationInst, '_close', rc);
```



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**

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

ISBN 1-58025-477-2

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, in 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, September 1999

SAS<sup>®</sup> 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. <sup>®</sup> indicates USA registration.

IBM<sup>®</sup>, AIX<sup>®</sup>, DB2<sup>®</sup>, OS/2<sup>®</sup>, OS/390<sup>®</sup>, RS/6000<sup>®</sup>, System/370<sup>™</sup>, and System/390<sup>®</sup> are registered trademarks or trademarks of International Business Machines Corporation. ORACLE<sup>®</sup> is a registered trademark or trademark of Oracle Corporation. <sup>®</sup> 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.