HP TP Desktop Connector for ACMS

Client Services Reference Manual

January 2006

This manual describes the services and commands needed to create and maintain TP Desktop Connector client programs that use the portable API.

Revision Update Information:  This is a revised manual.
Operating System:  OpenVMS Alpha Version 8.2
                   OpenVMS I64 Version 8.2-1
Software Version:  HP TP Desktop Connector for ACMS, Version 5.0

Hewlett-Packard Company
Palo Alto, California
Contents

Preface .............................................................. vii

1 Service Format
   1.1 Routine Names ............................................. 1–1
   1.2 Format ..................................................... 1–1
   1.3 Parameters ................................................. 1–1
     1.3.1 Type Entry ........................................... 1–2
     1.3.2 Access ............................................... 1–3
     1.3.3 Mechanism ........................................... 1–3
   1.4 Return Status ............................................. 1–4
   1.5 Session Environments ..................................... 1–4

2 TP Desktop Connector Portable API Client Services
   2.1 Summary of Portable API Client Services ................. 2–1
   2.2 Parameter Memory Allocation .............................. 2–2
   2.3 Nonblocking Service Usage ................................ 2–2
     2.3.1 Nonblocking and Blocking Restriction ............... 2–3
     2.3.2 Completion Routine Format ......................... 2–3
   2.4 Workspace Data Structures ............................... 2–4
     2.4.1 ACMSDI_WORKSPACE Structure and Initialization
           Macro .................................................. 2–4
     2.4.2 ACMSDI_WORKSPACE_OPT Structure ................. 2–6
     2.4.3 ACMSDI_WORKSPACE_BIND Structure ............... 2–7
     2.4.4 ACMSDI_FORM_RECORD_BIND Structure ............. 2–7
   2.5 ACMSDI_CALL_OPTION Union Structure .................. 2–8
     2.5.1 ACMSDI_OPTION Array ............................... 2–10
   2.6 acmsdi_call_task ......................................... 2–13
   2.7 acmsdi_cancel ............................................ 2–18
   2.8 acmsdi_complete_pp ...................................... 2–21
   2.9 acmsdi_dispatch_message ................................ 2–23
   2.10 acmsdi_return_pointer .................................. 2–25
3 Portable API Presentation Procedures

3.1 Summary of Portable API Presentation Procedures ................... 3–1
3.1.1 Return Status Values Expected from Presentation Procedures ........ 3–2
3.1.2 ACMSDI_FORM_RECORD Structure and Macro Call ........ 3–3
3.1.3 Prototypes and Code for Presentation Procedures and Version Routines .... 3–3
3.2 Parameter Memory Allocation ........................................ 3–3
3.3 Blocking and Nonblocking Usage .................................... 3–4
3.3.1 Presentation Procedures in a Nonblocking Environment .......... 3–4
3.3.2 Nonblocking and Blocking Restriction ............................ 3–4
3.4 acmsdi_disable ..................................................... 3–5
3.5 acmsdi_enable ..................................................... 3–6
3.6 acmsdi_read_msg .................................................. 3–9
3.7 acmsdi_receive ..................................................... 3–11
3.8 acmsdi_request ..................................................... 3–14
3.9 acmsdi_send ......................................................... 3–16
3.10 acmsdi_transceive .................................................. 3–19
3.11 acmsdi_write_msg .................................................. 3–23
3.12 Version-Checking Routines ........................................ 3–25
3.12.1 acmsdi_check_version ........................................ 3–25
3.12.2 acmsdi_get_version ........................................... 3–27

4 Forced Nonblocking Client Services

4.1 Summary of Forced Nonblocking Procedures .......................... 4–1
4.1.1 ACMSDI_FORM_RECORD_BIND Structure .......................... 4–2
4.1.2 ACMSDI_WORKSPACE_BIND Structure ............................ 4–3
4.2 acmsdi_complete_call .............................................. 4–4
4.3 acmsdi_bind_enable_args .......................................... 4–7
4.4 acmsdi_bind_msg ..................................................... 4–10
4.5 acmsdi_bind_receive_args ........................................ 4–13
4.6 acmsdi_bind_receive_recs ....................................... 4–15
4.7 acmsdi_bind_request_args ....................................... 4–17
4.8 acmsdi_bind_request_wksp ................................. 4–19
4.9 acmsdi_bind_send_args .......................................... 4–21
4.10 acmsdi_bind_send_recs ......................................... 4–23
4.11 acmsdi_bind_session_id ........................................... 4–25
4.12 acmsdi_bind_transceive_args .................................... 4–27
5 System Management Service on OpenVMS
   5.1 ACMSDI$GET_SUBMITTER_INFO ........................................... 5–2

6 Data Compression Monitor Commands
   6.1 EXIT ............................................ 6–2
   6.2 HELP ............................................ 6–3
   6.3 LIST ............................................ 6–4
   6.4 RENEW ........................................... 6–9
   6.5 SELECT ........................................... 6–10
   6.6 SET ............................................ 6–13
   6.7 SHOW ............................................ 6–15

A HP ACMS System Status Values

Index

Examples
   2–1 Workspace Structure Definition and Initialization ........ 2–5
   2–2 Passing Workspaces to a Procedure ......................... 2–5
   2–3 ACMSDI_WORKSPACE_OPT Type Definition ................... 2–6
   2–4 Passing Two Workspaces ................................... 2–6
   2–5 Initializing an Options List ............................... 2–11
   2–6 Dynamically Specifying a TCP/IP Port Identifier .......... 2–12
   3–1 Form Record Definition and Initialization Macro ......... 3–3
   4–1 Form Record Definition ...................................... 4–3
   4–2 Workspace Structure Definition ............................ 4–3
Figures
5–1 Submitter Item Descriptor Format .......................... 5–3

Tables
1–1 Services Description Parameters .......................... 1–1
1–2 Parameter Data Types ........................................ 1–2
1–3 Called Routine Access Methods .......................... 1–3
1–4 Parameter-Passing Mechanisms .......................... 1–4
1–5 Matrix of Services and Environments ................. 1–5
2–1 Summary of Portable API Client Services ............ 2–1
2–2 acmsdi_call_task Return Status Values ............... 2–16
2–3 acmsdi_cancel Return Status Values ................. 2–20
2–4 acmsdi_complete_pp Return Status Values .......... 2–22
2–5 acmsdi_dispatch_message Return Status Values ..... 2–23
2–6 acmsdi_sign_in Return Status Values ................. 2–28
2–7 acmsdi_sign_out Return Status Values ............... 2–30
3–1 Summary of Portable API Presentation Procedures ...... 3–1
4–1 Summary of Forced Nonblocking Procedures ........ 4–1
4–2 acmsdi_complete_call Return Status Values .......... 4–6
4–3 acmsdi_bind_enable_args Return Status Values ....... 4–9
4–4 acmsdi_bind_msg Return Status Values ................. 4–12
4–5 acmsdi_bind_receive_args Return Status Values .... 4–14
4–6 acmsdi_bind_receive_recs Return Status Values .... 4–16
4–7 acmsdi_bind_request_args Return Status Values .... 4–18
4–8 acmsdi_bind_request_wkspcs Return Status Values ... 4–20
4–9 acmsdi_bind_send_args Return Status Values ..... 4–22
4–10 acmsdi_bind_send_recs Return Status Values ....... 4–24
4–11 acmsdi_bind_session_id Return Status Values .... 4–26
4–12 acmsdi_bind_transceive_args Return Status Values ... 4–29
4–13 acmsdi_poll Return Status Values .................. 4–31
5–1 Submitter Information Item Codes ....................... 5–3
5–2 ACMSDI$GET_SUBMITTER_INFO Return Status Values ........................................ 5–6
A–1 ACMS System Status Values ................................ A–1
Preface

This manual provides reference information for the TP Desktop Connector client services, formerly known as the ACMS Desktop Portable API.

Intended Audience

This guide is intended for application programmers, application designers, and system managers.

Manual Structure

This manual has the following structure:

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 1</td>
<td>Explains the format of the reference information.</td>
</tr>
<tr>
<td>Chapters 2, 3, 4, and 5</td>
<td>Contain the reference information on TP Desktop Connector client services, presentation procedures, action routines, and the OpenVMS based system management service.</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Lists the data compression monitor commands.</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Lists the HP ACMS system status values that can be returned in the <code>err2</code> parameter.</td>
</tr>
</tbody>
</table>

Related Documents

For information on developing HP ACMS applications, refer to the following manuals:

- *HP TP Desktop Connector for ACMS Client Application Programming Guide*
  
  Provides information for designing, coding, and implementing a TP Desktop Connector solution.
• **HP TP Desktop Connector for ACMS Installation Guide**
  Provides the steps needed to install a TP Desktop Connector gateway on an OpenVMS system and the TP Desktop Connector software on the desktop client system.

• **HP TP Desktop Connector for ACMS Gateway Management Guide**
  Contains information about the system management and administration of the TP Desktop Connector gateway. It also includes information on the methodology of the use of network transports.

• **HP TP Desktop Connector for ACMS Getting Started**
  Provides a high-level discussion and examples of the activities to develop, install, and run a complete application.

If you are new to programming with ACMS software, HP recommends reading the following books before using the *HP TP Desktop Connector for ACMS* product:

• **HP ACMS for OpenVMS Writing Applications**
  Describes procedures to follow using the Application Development Utility (ADU).

• **HP ACMS for OpenVMS Writing Server Procedures**
  Describes how to write and debug procedures for ACMS applications. Also supplies reference information for application and system programming services.

For additional information on ACMS software, refer to the following manuals:

• **HP ACMS for OpenVMS Introduction**
  Describes basic concepts and terms concerning the ACMS environment.

• **HP ACMS for OpenVMS ADU Reference Manual**
  Describes the details of the syntax for the definitions you create and the commands you use to build the run-time components.

For information on OpenVMS programming tools, refer to this document:

• **Using HP DECset**
  Describes the OpenVMS programming environment, provides helpful hints about conducting a software project, and shows a case study of HP DECset tools. Provided with the HP DECset documentation set.
  The HP ACMS documentation also describes how you can use the HP DECset tools to create an effective development environment.
## Conventions

This guide uses the following conventions and symbols:

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TP Desktop Connector</strong></td>
<td>Refers to the <em>HP TP Desktop Connector</em> for ACMS software.</td>
</tr>
<tr>
<td><strong>User Input</strong></td>
<td>In examples, user input is highlighted with <strong>bold</strong> type.</td>
</tr>
<tr>
<td>$</td>
<td>The dollar sign indicates a generic command line prompt.</td>
</tr>
<tr>
<td></td>
<td>This prompt may be different on your system.</td>
</tr>
<tr>
<td>Return</td>
<td>A key name in a box indicates that you press that key on the keyboard.</td>
</tr>
<tr>
<td><strong>Ctrl</strong>/x</td>
<td>Press the Ctrl (control) key and hold it down while pressing the specified key (indicated here by x).</td>
</tr>
<tr>
<td><strong>WORD</strong></td>
<td>Uppercase text indicates OpenVMS data types, commands, keywords, logical names, and routines or services; C files and data structures; Microsoft Windows data structures; or HyperCard data types.</td>
</tr>
<tr>
<td><strong>word</strong></td>
<td>In format descriptions, lowercase words indicate parameters, variables, services, or procedures.</td>
</tr>
</tbody>
</table>
**italics**  
*Italics* are used for emphasis and for parameters in text.  
Titles of manuals are also italicized.

[]  
In format descriptions, square brackets surround a choice of options; select none, one, several, or all of the choices.

.  
.  
A vertical ellipsis in an example means that information not directly related to the example has been omitted.

**Windows**  
When used alone, Windows indicates any supported member of the family of Microsoft Windows operating systems. Where necessary, specific Windows operating systems are mentioned. For a list of Microsoft Windows operating systems supported by the TP Desktop Connector product, see the product’s *Software Product Description* (SPD).
This chapter describes the format and elements of the service descriptions provided in following chapters. This chapter also provides a list of the services and the appropriate session environments in which each service may be used.

1.1 Routine Names

The TP Desktop Connector service names and OpenVMS action routines are shown in C-language format. The OpenVMS system management services are shown in the OpenVMS services format.

1.2 Format

The format section describes the C functions as they are declared for the portable API in the include file ACMSDI.H in the ACMSDI$COMMON directory.

Square brackets ([[]]) indicate optional parameters in the call.

1.3 Parameters

This section contains details about each parameter listed in the format section. Parameters appear in the order in which they are shown in the format. The format shown in Table 1–1 describes each parameter.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Data type of the parameter</td>
</tr>
<tr>
<td>Access</td>
<td>Method by which the called routine accesses the parameter</td>
</tr>
<tr>
<td>Mechanism</td>
<td>Method by which a parameter is passed to the called routine</td>
</tr>
</tbody>
</table>

The parameters section additionally contains a sentence or two describing the purpose of the parameter.
### 1.3.1 Type Entry

Table 1–2 lists the C-language data types used in the TP Desktop Connector services.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_CALL_ID</td>
<td>Identification returned by the acmsdi_call_task service</td>
</tr>
<tr>
<td>ACMSDI_FORM_RECORD</td>
<td>Structure defined in the ACMSDL.H include file (see Section 3.1.2)</td>
</tr>
<tr>
<td>ACMSDI_FORM_RECORD_BIND</td>
<td>Structure defined in the ACMSDL.H and ACMSDL.BAS include files (see Section 4.1.1)</td>
</tr>
<tr>
<td>ACMSDI_FORMS_SESSION_ID</td>
<td>Structure defined in the ACMSDL.H include file (see Section 3.5)</td>
</tr>
<tr>
<td>ACMSDI_OPTION</td>
<td>Union to specify sign-in options (see Section 2.11)</td>
</tr>
<tr>
<td>ACMSDI_CALL_OPTIONS</td>
<td>Union to specify call task options</td>
</tr>
<tr>
<td>ACMSDI_SUBMITTER_ID</td>
<td>Structure defined in the ACMSDL.H include file (see Section 2.11)</td>
</tr>
<tr>
<td>ACMSDI_WORKSPACE</td>
<td>Array of structures defined in the ACMSDL.H include file to pass workspaces between the desktop system and the TP Desktop Connector gateway (see Section 2.4)</td>
</tr>
<tr>
<td>ACMSDI_WORKSPACE_BIND</td>
<td>Structure defined in the ACMSDL.H and ACMSDL.BAS include files (see Section 4.1.2)</td>
</tr>
<tr>
<td>ACMSDI_WORKSPACE_OPT</td>
<td>Array of structures defined in the ACMSDL.H include file to pass unidirectional workspaces between the desktop system and the TP Desktop Connector server</td>
</tr>
<tr>
<td>char *</td>
<td>Array of unsigned 8-bit integers</td>
</tr>
<tr>
<td>character string descriptor</td>
<td>Address of an OpenVMS string descriptor pointing to the character string to be passed</td>
</tr>
<tr>
<td>function address</td>
<td>Address of a function that complies with the prototype in ACMSDL.H for the completion routine</td>
</tr>
<tr>
<td>int</td>
<td>32-bit signed integer</td>
</tr>
<tr>
<td>long</td>
<td>Synonym for long int</td>
</tr>
<tr>
<td>long int</td>
<td>32-bit signed integer</td>
</tr>
</tbody>
</table>

(continued on next page)
1.3.2 Access

Access describes the way in which the called routine accesses the data specified by the parameter. The access methods are described in Table 1–3.

Table 1–3 Called Routine Access Methods

<table>
<thead>
<tr>
<th>Access Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Data needed by the called routine to perform its operation is read but not returned.</td>
</tr>
<tr>
<td>Write</td>
<td>Data that the called routine returns to the calling routine is written into a location accessible to the calling routine.</td>
</tr>
<tr>
<td>Modify</td>
<td>Data is both read and returned by the called routine; input data specified by the parameter is overwritten.</td>
</tr>
</tbody>
</table>

1.3.3 Mechanism

The parameter-passing mechanism is the way in which a parameter specifies the data to be used by the called routine. The passing mechanisms are described in Table 1–4.
### Table 1–4 Parameter-Passing Mechanisms

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>By value</td>
<td>The parameter contains a copy of the data to be used by the routine.</td>
</tr>
<tr>
<td>By reference</td>
<td>The parameter contains the address of the data to be used by the routine. The parameter is a pointer to the data. Because C supports only call by value, write parameters other than arrays and structures must be passed as pointers. References to names of arrays and structures are converted by the compiler to pointer expressions.</td>
</tr>
</tbody>
</table>

For information on whether the caller or the called routine allocates memory, see the discussions of the individual platforms.

### 1.4 Return Status

Each service returns a status value defined as follows:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>long int</td>
</tr>
<tr>
<td>OpenVMS</td>
<td>long int</td>
</tr>
<tr>
<td>Tru64 UNIX</td>
<td>long int</td>
</tr>
</tbody>
</table>

Only the status codes defined in the related reference sections are valid in the TP Desktop Connector client services. The definitions for the return status values are in include files as follows:

<table>
<thead>
<tr>
<th>Type of Services</th>
<th>Include File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portable client services</td>
<td>ACMSDI$COMMON:ACMSDI.H</td>
</tr>
</tbody>
</table>

### 1.5 Session Environments

Client services can be used in three different session environments, blocking, nonblocking, and forced nonblocking. In a blocking environment, service routines are completed in one procedure. In a nonblocking environment, service routines return control to the desktop client program as soon as a request is sent and then call the appropriate completion routine when the request is completed or call the appropriate presentation procedure when an exchange step is detected.
In a forced nonblocking environment, service routines provide a method of polling that is used to determine the type of message sent from the back-end server. This message type may then be used to determine the appropriate action (for example, process the call completion or exchange step). The forced nonblocking software provides additional routines to access call completion and exchange step arguments. These session environments are explained in more depth in Chapter 2 and in *HP TP Desktop Connector for ACMS Client Application Programming Guide*.

Table 1–5 lists the services and indicates the session environments in which you can use each call.

### Table 1–5 Matrix of Services and Environments

<table>
<thead>
<tr>
<th>Service</th>
<th>Availability within Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Blocking</td>
</tr>
<tr>
<td>acmsdi_call_task</td>
<td>yes</td>
</tr>
<tr>
<td>See description in Section 2.6</td>
<td></td>
</tr>
<tr>
<td>acmsdi_cancel</td>
<td>-</td>
</tr>
<tr>
<td>See description in Section 2.7</td>
<td></td>
</tr>
<tr>
<td>acmsdi_complete_pp</td>
<td>-</td>
</tr>
<tr>
<td>See description in Section 2.8</td>
<td></td>
</tr>
<tr>
<td>acmsdi_dispatch_message</td>
<td>-</td>
</tr>
<tr>
<td>See description in Section 2.9</td>
<td></td>
</tr>
<tr>
<td>acmsdi_return_pointer</td>
<td>yes</td>
</tr>
<tr>
<td>See description in Section 2.10</td>
<td></td>
</tr>
<tr>
<td>acmsdi_sign_in</td>
<td>yes</td>
</tr>
<tr>
<td>See description in Section 2.11</td>
<td></td>
</tr>
<tr>
<td>acmsdi_sign_out</td>
<td>yes</td>
</tr>
<tr>
<td>See description in Section 2.12</td>
<td></td>
</tr>
<tr>
<td>acmsdi_poll</td>
<td>-</td>
</tr>
<tr>
<td>See description in Section 4.13</td>
<td></td>
</tr>
<tr>
<td>acmsdi_complete_call</td>
<td>-</td>
</tr>
<tr>
<td>See description in Section 4.2</td>
<td></td>
</tr>
<tr>
<td>acmsdi_bind_enable_args</td>
<td>-</td>
</tr>
<tr>
<td>See description in Section 4.3</td>
<td></td>
</tr>
<tr>
<td>acmsdi_bind_send_args</td>
<td>-</td>
</tr>
<tr>
<td>See description in Section 4.9</td>
<td></td>
</tr>
</tbody>
</table>

(continued on next page)
Table 1–5 (Cont.) Matrix of Services and Environments

<table>
<thead>
<tr>
<th>Service</th>
<th>Availability within Environment</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Blocking</td>
<td>Nonblocking</td>
<td>Forced Nonblocking</td>
</tr>
<tr>
<td>acmadi_bind_receive_args</td>
<td></td>
<td>-</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>See description in Section 4.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acmadi_bind_transceive_args</td>
<td></td>
<td>-</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>See description in Section 4.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acmadi_bind_msg</td>
<td></td>
<td>-</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>See description in Section 4.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acmadi_bind_request_args</td>
<td></td>
<td>-</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>See description in Section 4.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acmadi_bind_session_id</td>
<td></td>
<td>-</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>See description in Section 4.11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acmadi_bind_send_recs</td>
<td></td>
<td>-</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>See description in Section 4.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acmadi_bind_receive_recs</td>
<td></td>
<td>-</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>See description in Section 4.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acmadi_bind_request_wksps</td>
<td></td>
<td>-</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>See description in Section 4.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Callbacks**

<table>
<thead>
<tr>
<th>Service</th>
<th>Availability within Environment</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Blocking</td>
<td>Nonblocking</td>
<td>Forced Nonblocking</td>
</tr>
<tr>
<td>acmadi_disable</td>
<td></td>
<td>-</td>
<td>yes</td>
<td>-</td>
</tr>
<tr>
<td>See description in Section 3.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acmadi_enable</td>
<td></td>
<td>-</td>
<td>yes</td>
<td>-</td>
</tr>
<tr>
<td>See description in Section 3.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acmadi_read_msg</td>
<td></td>
<td>-</td>
<td>yes</td>
<td>-</td>
</tr>
<tr>
<td>See description in Section 3.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acmadi_receive</td>
<td></td>
<td>-</td>
<td>yes</td>
<td>-</td>
</tr>
<tr>
<td>See description in Section 3.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acmadi_request</td>
<td></td>
<td>-</td>
<td>yes</td>
<td>-</td>
</tr>
<tr>
<td>See description in Section 3.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>acmadi_send</td>
<td></td>
<td>-</td>
<td>yes</td>
<td>-</td>
</tr>
<tr>
<td>See description in Section 3.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(continued on next page)
### Table 1–5 (Cont.) Matrix of Services and Environments

<table>
<thead>
<tr>
<th>Callbacks</th>
<th>Availability within Environment</th>
<th>Forced Nonblocking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Blocking</td>
<td>Nonblocking</td>
</tr>
<tr>
<td>acmsdi_transceive</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>See description in Section 3.10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>acmsdi_write_msg</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>See description in Section 3.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>acmsdi_check_version</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>See description in Section 3.12.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>acmsdi_get_version(back end)</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>See description in Section 3.12.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This chapter describes the HP TP Desktop Connector portable API client services available on the following desktop systems:

- Microsoft Windows
- OpenVMS
- HP Tru64 UNIX

2.1 Summary of Portable API Client Services

Similar to the HP ACMS Service Interface (SI) routines provided on the OpenVMS host, the TP Desktop Connector portable API client services allow you to write a desktop client program on desktop systems without extensive knowledge of network communications. Table 2–1 summarizes the TP Desktop Connector portable API client services.

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acmsdi_call_task</td>
<td>Sends a request to the TP Desktop Connector gateway to start a task in a ACMS application. The TP Desktop Connector client service is either blocking or nonblocking. Exchange step processing during the task is handled by the TP Desktop Connector gateway calling customer-written generic presentation procedures in the desktop client program.</td>
</tr>
<tr>
<td>acmsdi_cancel</td>
<td>Used by nonblocking services only. Called by a desktop application to cancel a currently active ACMS task.</td>
</tr>
</tbody>
</table>

(continued on next page)
Table 2–1 (Cont.) Summary of Portable API Client Services

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acmsdi_complete_pp</td>
<td>Used by nonblocking environments only. Sends a response from a presentation procedure request to the TP Desktop Connector gateway.</td>
</tr>
<tr>
<td>acmsdi_dispatch_message</td>
<td>Used by nonblocking environments only. Checks for and processes messages from the TP Desktop Connector gateway. If no messages have been received from the gateway, acmsdi_dispatch_message returns immediately.</td>
</tr>
<tr>
<td>acmsdi_return_pointer</td>
<td>Used by client programs written in Microsoft Visual Basic to create the workspace array for ACMS_CALL_TASK. Also used in the forced nonblocking environment to obtain reference pointers.</td>
</tr>
<tr>
<td>acmsdi_sign_in</td>
<td>Requests the TP Desktop Connector gateway to sign a user running a desktop client program in to an ACMS system.</td>
</tr>
<tr>
<td>acmsdi_sign_out</td>
<td>Requests the TP Desktop Connector gateway to sign a desktop client program out of a ACMS system.</td>
</tr>
</tbody>
</table>

These calls use the C-language argument-passing standards. Character strings are NULL-terminated and passed by reference. Workspaces are passed as structures composed of a length and a pointer field.

2.2 Parameter Memory Allocation

The caller of a TP Desktop Connector client service or a presentation procedure is responsible for allocating the memory for the parameters of that routine. For calls to the TP Desktop Connector client services, the desktop client program must allocate the memory for all parameters passed in, for example, submitter_id and call_context. For calls to the presentation procedures, the TP Desktop Connector client services allocate memory for all the parameters passed and for all workspaces.

2.3 Nonblocking Service Usage

The acmsdi_sign_in, acmsdi_call_task, and acmsdi_sign_out services can be either blocking, nonblocking, or forced nonblocking. If the desktop client program supplies the completion_routine parameter to the TP Desktop Connector client service, the service behaves in the nonblocking fashion. The TP Desktop Connector client service returns control to the desktop client program as soon as a request is sent to the TP Desktop Connector gateway. If the request is sent to the gateway successfully, the TP Desktop Connector...
client service returns the ACMSDI_PENDING status. If a status other than ACMSDI_PENDING is returned, the completion routine is not called.

If nonblocking calls are active, use the acmsdi_dispatch_message service to poll for responses from the TP Desktop Connector gateway. When a response is received, acmsdi_dispatch_message calls the appropriate customer-supplied completion routine. If the desktop client program supplies the completion_status parameter on the initial TP Desktop Connector client service call, the TP Desktop Connector client services set the completion_status to the final completion status for the service and immediately call the completion routine. See HP TP Desktop Connector for ACMS Client Application Programming Guide for descriptions and examples.

The forced nonblocking services extend the portable API to support both exchange steps and nonblocking execution of task calls for development tools that do not support pointer data types or whose memory management routines relocate data. You can specify a forced nonblocking session with the acmsdi_sign_in service by using the ACMSDI_OPTION, ACMSDI_OPT_NONBLK. Do not specify a completion routine in a forced nonblocking session as this will result in an error. See Chapter 4 for more information.

2.3.1 Nonblocking and Blocking Restriction

All calls using the same desktop client program and TP Desktop Connector gateway connection must be either blocking, nonblocking, or forced nonblocking. These types of service calls cannot be mixed for a desktop client program and TP Desktop Connector gateway pair. See Table 1–5 for the list of service calls available for each type of session. If a desktop client program connects to two different TP Desktop Connector gateways, it can mix service call types, using blocking calls to interact with one TP Desktop Connector gateway and nonblocking calls to interact with the other TP Desktop Connector gateway.

2.3.2 Completion Routine Format

For nonblocking service requests, the acmsdi_dispatch_message service calls the customer-supplied completion routine when a response is received from the TP Desktop Connector gateway. The completion routine has the following format:

```c
void completion_routine (call_context)
```

Parameters
call_context
Type: void *
Access: read
Mechanism: by value
Supplies application-specific context to the completion routine. If specified on
acmsdi_call_task, acmsdi_sign_in, acmsdi_cancel, or acmsdi_sign_out service,
the call_context is passed by the TP Desktop Connector client services to the
completion routine.

Return Status
The customer-supplied completion routine does not return a status value.

2.4 Workspace Data Structures
This section describes the following workspace data structures:

    ACMSDI_WORKSPACE
    ACMSDI_WORKSPACE_OPT
    ACMSDI_WORKSPACE_BIND
    ACMSDI_FORM_RECORD_BIND

2.4.1 ACMSDI_WORKSPACE Structure and Initialization Macro
Defined in the ACMSDI.H file, the ACMSDI_WORKSPACE type declares
workspaces passed to tasks using the acmsdi_call_task service and workspaces
passed from tasks to acmsdi_request presentation procedures.

The code in Example 2–1 defines the ACMSDI_WORKSPACE type and
an ACMSDI_INIT_WORKSPACE macro used to initialize the workspace
structure.
Example 2–1 Workspace Structure Definition and Initialization

typedef struct {
    unsigned int length; /** length of workspace **/
    void *data;    /** pointer to workspace **/
} ACMSDI_WORKSPACE;
.
.
#define ACMSDI_INIT_WORKSPACE(_wksp, _rec)\
{\
    _wksp.length = sizeof(_rec);\
    _wksp.record = &(_rec);\
}
.
.
To pass more than one workspace to a procedure, use an array of the ACMSDI_WORKSPACE structures. Example 2–2 passes two workspaces.

Example 2–2 Passing Workspaces to a Procedure

ACMSDI_WORKSPACE wksp_array[2];

struct {
    char ctrl_key[5];
    char error_message[80];
} control_wksp;

struct {
    int id_number;
    char first_name[15];
    char last_name[25];
} employee_record;

ACMSDI_INIT_WORKSPACE (wksp_array[0], control_wksp);
ACMSDI_INIT_WORKSPACE (wksp_array[1], employee_record);

The array wksp_array is defined with two elements of type ACMSDI_WORKSPACE. The structure definitions control_wksp and employee_record define the elements of the array. The two macro ACMSDI_INIT_WORKSPACE calls initialize the array of structures.
2.4.2 ACMSDI_WORKSPACE_OPT Structure

The ACMSDI.H file contains the definition of the ACMSDI_WORKSPACE_OPT type you use to declare workspaces passed to tasks using the ACMSDI_CALL_TASK service. You can use ACMSDI_WORKSPACE_OPT instead of ACMSDI_WORKSPACE. Only task calls that use the ACMSDI_TASK_OPTIONS flag to indicate unidirectional workspaces can use this structure. Example 2–3 shows the ACMSDI_WORKSPACE_OPT type definition and the definition of a macro to initialize the workspace structure.

Example 2–3 ACMSDI_WORKSPACE_OPT Type Definition

```
#define ACMSDI_ACCESS_READ '1' /* read-only access */
#define ACMSDI_ACCESS_WRITE '2' /* write-only access */
#define ACMSDI_ACCESS_MODIFY '3' /* modify (read and write) */

typedef char ACMSDI_ACCESS_TYPE;
typedef struct {
    unsigned int length;
    ACMSDI_ACCESS_TYPE access;
    void *data;
} ACMSDI_WORKSPACE_OPT;

#define ACMSDI_INIT_WORKSPACE_OPT(_wksp, _rec, _access)
{ _wksp.length = sizeof(_rec);
  _wksp.access = _access;
  _wksp.data = &(_rec);
}
```

To pass more than one workspace to a procedure, use an array of the type ACMSDI_WORKSPACE_OPT. Example 2–4 passes two workspaces.

Example 2–4 Passing Two Workspaces

```
ACMSDI_WORKSPACE_OPT wksp_array[2];
struct {
    char ctrl_key[5];
    char error_message[80];
} control_wksp;
```

(continued on next page)
Example 2–4 (Cont.) Passing Two Workspaces

```c
struct {
    int id_number;
    char first_name[15];
    char last_name[25];
} employee_record;

ACMSDI_INIT_WORKSPACE_OPT (wksp_array[0], control_wksp, ACMSDI_ACCESS_READ);
ACMSDI_INIT_WORKSPACE_OPT (wksp_array[1], employee_record, ACMSDI_ACCESS_WRITE);
```

### 2.4.3 ACMSDI_WORKSPACE_BIND Structure

The ACMSDI_WORKSPACE_BIND structure locates workspace buffers and specifies the sizes of workspaces during acmsdi_bind_request_wksps operations. Like the ACMSDI_FORM_RECORD_BIND structure, the ACMSDI_WORKSPACE_BIND structure contains a field where the length of the TDMS exchange step workspace is returned. If the length differs from the buffer length, TP Desktop Connector truncates the resultant workspaces or buffers are not completely filled.

The following example shows the C language definition of this structure as it appears in the acmsdi.h file:

```c
typedef struct {
    unsigned int buffer_len; /* length of caller’s buffer */
    unsigned int wksp_len; /* actual length of the workspace */
    void *data;
} ACMSDI_WORKSPACE_BIND;
```

### 2.4.4 ACMSDI_FORM_RECORD_BIND Structure

The ACMSDI_FORM_RECORD_BIND structure locates form record buffers and specifies their sizes during acmsdi_bind_send_recs and acmsdi_bind_receive_recs operations. ACMSDI_FORM_RECORD_BIND serves the same purpose as ACMSDI_FORM_RECORD with one additional feature. It contains an additional field, rec_len, with which the TP Desktop Connector client services return the actual length of the form record as it is returned from the back-end application. You can compare this length against the client application buffer length to see if the buffer is large enough, too large, or exactly the right size to contain the form record. If the buffer size is too small, the form record is truncated to fit the buffer. If the buffer size is too large, the buffer is not completely filled.
You can use the ACMSDI_FORM_RECORD_BIND structure to locate send control text and receive control text buffers. Both acmsdi_bind_send_args and acmsdi_bind_receive_args services contain arguments to specify whether or not to transfer control text. If you specify to transfer control text, the following rules apply:

- ACMSDI_FORM_RECORD_BIND structure for the control text must be the first one in the array of such structures passed on the call.
- After the call completes, the record length field (rec_len) contains the send control text count or the receive control text count instead of the length of the record.

The following example shows the C language definition of this structure as it appears in the acmsdi.h file:

```c
typedef struct {
    unsigned int buffer_len; /* length of caller’s record buffer */
    unsigned int rec_len;   /* actual length of the form record */
    void        *data_record;
    unsigned int shadow_buffer_len; /* length of caller’s shadow buffer */
    unsigned int shadow_rec_len;   /* actual length of shadow record */
    void        *shadow_record;
} ACMSDI_FORM_RECORD_BIND;
```

### 2.5 ACMSDI_CALL_OPTION Union Structure

ACMSDI_CALL_OPTION union is a parameter that is passed to the ACMSDI_CALL_TASK service to enable TP Desktop Connector functions, such as optimizing unidirectional workspace traffic on the call to the acmsdi_call_task client service. The include file ACMSDI.H contains the definition for the ACMSDI_CALL_OPTION union.

ACMSDI_CALL_OPTION contains several structures with the option variables, whose values determine the type of option being selected. Specify the values for the option variable using the following constants defined in the ACMSDI.H include file:
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_CALL_OPT_END_LIST</td>
<td>Ends options list</td>
</tr>
<tr>
<td>ACMSDI_CALL_OPT_OPTIMIZE_WKPS</td>
<td>Enables unidirectional workspace optimization</td>
</tr>
<tr>
<td>ACMSDI_CALL_OPT_ENABLE</td>
<td>Pointer to enable function</td>
</tr>
<tr>
<td>ACMSDI_CALL_OPT_DISABLE</td>
<td>Pointer to disable function</td>
</tr>
<tr>
<td>ACMSDI_CALL_OPT_SEND</td>
<td>Pointer to send function</td>
</tr>
<tr>
<td>ACMSDI_CALL_OPT_RECEIVE</td>
<td>Pointer to receive function</td>
</tr>
<tr>
<td>ACMSDI_CALL_OPT_TDMS_READ</td>
<td>Pointer to TDMS read function</td>
</tr>
<tr>
<td>ACMSDI_CALL_OPT_TDMS_WRITE</td>
<td>Pointer to TDMS write function</td>
</tr>
<tr>
<td>ACMSDI_CALL_OPT_TRANSCEIVE</td>
<td>Pointer to transceive function</td>
</tr>
<tr>
<td>ACMSDI_CALL_OPT_REQUEST</td>
<td>Pointer to TDMS request function</td>
</tr>
<tr>
<td>ACMSDI_CALL_OPT_CHECK_VERSION</td>
<td>Version checking routine</td>
</tr>
<tr>
<td>ACMSDI_CALL_OPT_PASS_TID</td>
<td>TID of distributed transaction</td>
</tr>
<tr>
<td>ACMSDI_CALL_OPT_COMPRESS_WKPS</td>
<td>Activate workspace compression</td>
</tr>
</tbody>
</table>

To select options:

1. Declare an array of at least two elements of the type ACMSDI_CALL_OPTION.

2. Specify in the option variable the name for the structure being used.

3. Specify the address for the malloc routine or password expiring buffer, if these options are being used.

4. End an options list by assigning ACMSDI_CALL_OPT_END_LIST to the option variable in the last array element.

The following example shows the initialization of an options list used to enable unidirectional workspace handling:

```c
ACMSDI_CALL_OPTION call_options[2];
    call_options[0].option = ACMSDI_CALL_OPT_OPTIMIZE_WKPS;
    call_options[1].option = ACMSDI_CALL_OPT_END_LIST;
```
Use the ACMSDI_CALL_OPT_OPTIMIZE_WKSPS option and the ACMSDI_WORKSPACE_OPT type definition together to optimize unidirectional workspace traffic. Do not use one without the other. The acmsdi_call_task client service uses the presence or absence of the workspace optimization option to decide which data type has been passed in the workspaces argument. Using either one without the other produces unpredictable results.

2.5.1 ACMSDI_OPTION Array

ACMSDI_OPTION array is a parameter that is passed to the ACMSDI_SIGN_IN service to enable TP Desktop Connector functions, such as enabling password expiration checking on the call to acmsdi_call_task client service. The include file ACMSDI.H contains the definition for the ACMSDI_OPTION array.

The ACMSDI_OPTION array is a union containing multiple structures and an option variable, the value of which defines the type of option being selected. Specify the values for the option variable using the following constants defined in the include file ACMSDI.H:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_OPT_CHECK_VERSION</td>
<td>Enables version checking</td>
</tr>
<tr>
<td>ACMSDI_OPT_COMMID</td>
<td>Supplies communications device id or TCP/IP comm port</td>
</tr>
<tr>
<td>ACMSDI_OPT_END_LIST</td>
<td>Ends options list</td>
</tr>
<tr>
<td>ACMSDI_OPT_FREE_ROUTINE</td>
<td>Enables user-defined memory deallocation</td>
</tr>
<tr>
<td>ACMSDI_OPT_MALLOC_ROUTINE</td>
<td>Enables user-defined memory allocation</td>
</tr>
<tr>
<td>ACMSDI_OPT_NONBLK</td>
<td>Enables a forced nonblocking session</td>
</tr>
<tr>
<td>ACMSDI_OPT_PWD_EXPIRING</td>
<td>Enables checking for passwords that are about to expire</td>
</tr>
</tbody>
</table>
To select options:

1. Declare an array of at least two elements of the type ACMSDI_OPTION.
2. Specify in the option variable the name tag for the structure being used.
3. End an options list by assigning ACMSDI_OPT_END_LIST to the option variable in the last array element.

Example 2–5 initializes an options list to enable version checking, user-defined memory allocation, and password expiration checking.

Example 2–5 Initializing an Options List

```c
void *my_malloc_routine(int size);
long pwd_exp_buffer;
void my_free_routine(void *ptr);
ACMSDI_OPTION options[5];

options[0].option = ACMSDI_OPT_CHECK_VERSION;
options[1].option = ACMSDI_OPT_MALLOC_ROUTINE;
options[1].malloc_routine.address = ... = &pwd_exp_buffer;
options[2].option = ACMSDI_OPT_FREE_ROUTINE;
options[2].free_routine.address = my_free_routine;
options[3].option = ACMSDI_OPT_PWD_EXPIRING;
options[3].pwd_expiring_hrs.address = &pwd_exp_buffer;
options[4].option = ACMSDI_OPT_END_LIST;
```

You can provide the TCP/IP port number during sign-in by using the ACMSDI_OPTSCOMMID option. Example 2–6 shows how to do this in C.

```
                       Note

This option is usable with forced nonblocking calls only.
```

If the environmental variable ACMSDI_TCPIP_PORT_host_node is defined, the option specified on the acmsdi_sign_in call takes precedence. If neither the environmental variable nor the sign-in option is specified, the default TCP/IP port number, 1023, is used.
Example 2–6  Dynamically Specifying a TCP/IP Port Identifier

```c
int status;
ACMSDI_SUBMITTER_ID subm_id;
long tcpip_port = 1000;
ACMSDI_OPTION options[2];
options[0].option = ACMSDI_OPT_COMMID;
options[0].CommID = tcpip_port;
options[1].option = ACMSDI_OPT_END_LIST;

status = acmsdi_sign_in ("N2001", /* ACMS Desktop Gateway node */
    "HAL", /* username */
    "HELLO_DAVE", /* password */
    options, /* sign in options */
    &subm_id, /* submitter id */
    0, 0, 0);
```
2.6 acmsdi_call_task

TP Desktop Connector client programs call this service to execute a task in an ACMS application.

**Format**

```
acmsdi_call_task (submitter_id,
[call_options],
task_name,
application_name,
selection_string,
status_message,
workspace_count,
[workspaces],
[call_id],
[completion_status],
[completion_routine], ¹
[call_context])
```

**Parameters**

- **submitter_id**
  - Type: ACMSDI_SUBMITTER_ID
  - Access: read
  - Mechanism: by reference
  - The submitter_id returned by the acmsdi_sign_in service.

- **call_options**
  - Type: ACMSDI_CALL_OPTION
  - Access: read
  - Mechanism: by reference
  - An array of ACMSDI_CALL_OPTION elements that either enables unidirectional workspace optimization or defines presentation procedure addresses. The include file ACMSDLH contains the definition for the ACMSDI_CALL_OPTION type. If you use the options array to enable unidirectional workspaces, use the ACMSDI_WORKSPACE_OPT type in the workspace list. See Section 2.4.2 and Section 2.5 for more information.

¹ For nonblocking only, see Section 2.3.
acmsdi_call_task

**task_name**
Type: `char *`
Access: **read**
Mechanism: **by reference**
The name of the task to execute. Maximum length is 31.

**application_name**
Type: `char *`
Access: **read**
Mechanism: **by reference**
The specification of a ACMS application in which the task resides. The application name must be a valid application specification on the submitter node. It can take the form NODE::APPLICATION, or can be specified using a logical name that is translated by the ACMS Central Controller (ACC) on the submitter node. Maximum length is 80.

**selection_string**
Type: `char *`
Access: **read**
Mechanism: **by reference**
Used by the desktop client program to pass additional information to the task. Maximum length is 256.

**status_message**
Type: `char *`
Access: **write**
Mechanism: **by reference**
A buffer to receive the message text associated with the task completion status. The message text returned is either the text associated with a TP Desktop Connector error or the message text returned from a ACMS application. Required length is 80.

---
**Caution**

If the full space is not allocated, the TP Desktop Connector client services write past the end of the allocated string and can cause the application to fail. Ensure that the desktop client program allocates the required length of space.
**acmsdi_call_task**

**workspace_count**
Type: *long int*
Access: *read*
Mechanism: *by value*
The decimal number of workspaces being passed to the task.

**workspaces**
Type: *ACMSDI_WORKSPACE* or *ACMSDI_WORKSPACE_OPT* array
Access: *read/write*
Mechanism: *by reference*
One or more optional workspaces to be passed to the task. You need to typecast your array to void *. The workspaces must be specified in the same order as they are declared in the task definition, and must match the number specified in the *workspace_count* parameter. If you use the ACMSDI_WORKSPACE_OPT type, you must set the call_options parameter to allow unidirectional workspaces.

**call_id**
Type: *ACMSDI_CALL_ID*
Access: *write*
Mechanism: *by reference*
A structure defined in the ACMSDI.H include file into which the acmsdi_call_task service writes a newly created call identification, a handle used by the TP Desktop Connector client services to identify an active call for a submitter.

**completion_status**
Type: *int*
Access: *write*
Mechanism: *by reference*
The final status of the TP Desktop Connector client service. In the blocking environment, the *completion_status* parameter is set to zero when the service starts successfully.

When the service completes, the *completion_status* parameter contains the final status. See Table 2–2 for the list of return status values.

When a task is canceled, the TP Desktop Connector gateway reports a specific error, where possible. If the gateway cannot convert a ACMS error to a specific TP Desktop Connector status, it returns ACMSDI_TASK_FAILED to the desktop client program.
acmsdi_call_task

completion_routine
Type: function address
Access: read
Mechanism: by value
Address of a function to be called when the service completes. The completion_routine is called by the acmsdi_dispatch_message service when the "End of Task" message is received from the TP Desktop Connector gateway.

call_context
Type: void *
Access: read
Mechanism: by value
Optional parameter passed to presentation procedures and completion routines to identify the call. Use this parameter to supply an application-specific context for the call.

Return Status
The status values returned by the acmsdi_call_task service are listed in Table 2–2.

Table 2–2 acmsdi_call_task Return Status Values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_APPLDEAD</td>
<td>Application stopped unexpectedly.</td>
</tr>
<tr>
<td>ACMSDI_CALLACTV</td>
<td>Call is already active.</td>
</tr>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>Insufficient parameters.</td>
</tr>
<tr>
<td>ACMSDI_INTERNAL</td>
<td>Internal TP Desktop Connector error.</td>
</tr>
<tr>
<td>ACMSDI_INVOPTION</td>
<td>Invalid item in options list.</td>
</tr>
<tr>
<td>ACMSDI_INVSUBID</td>
<td>Invalid or obsolete submitter identification.</td>
</tr>
<tr>
<td>ACMSDI_MIXEDMODE</td>
<td>All calls on a connection must be either blocking or nonblocking.</td>
</tr>
<tr>
<td>ACMSDI_NOMEMORY</td>
<td>Insufficient memory to complete requests.</td>
</tr>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_NOSUCH_APPL</td>
<td>Application not found.</td>
</tr>
</tbody>
</table>

(continued on next page)

1 For nonblocking only, see Section 2.3. Not applicable to forced nonblocking and will cause an error if supplied.
Table 2–2 (Cont.) acmsdi_call_task Return Status Values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_NOSUCH_TASK</td>
<td>Task not found.</td>
</tr>
<tr>
<td>ACMSDI_OPRCANCELLED</td>
<td>Operator canceled task.</td>
</tr>
<tr>
<td>ACMSDI_PENDING</td>
<td>Successful operation pending nonblocking completion. The final status is in</td>
</tr>
<tr>
<td></td>
<td>the completion status parameter.</td>
</tr>
<tr>
<td>ACMSDI_SECCHK</td>
<td>Task security check failed.</td>
</tr>
<tr>
<td>ACMSDI_SIGNINACTV</td>
<td>Request is invalid while the sign-in is active.</td>
</tr>
<tr>
<td>ACMSDI_SIGNOUTACTV</td>
<td>Request is invalid while the sign-out is active.</td>
</tr>
<tr>
<td>ACMSDI_SRVDEAD</td>
<td>Node name is invalid, or TP Desktop Connector gateway is not running on the</td>
</tr>
<tr>
<td></td>
<td>specified node, or the network link terminated.</td>
</tr>
<tr>
<td>ACMSDI_TASK_ABORT</td>
<td>Task completed abnormally.</td>
</tr>
<tr>
<td>ACMSDI_TASK_CANCELLED</td>
<td>Task was canceled.</td>
</tr>
<tr>
<td>ACMSDI_TASK_FAILED</td>
<td>Task failed during execution.</td>
</tr>
<tr>
<td>ACMSDI_TASK_SP_DIED</td>
<td>Task was canceled when TP Desktop Connector gateway process died.</td>
</tr>
</tbody>
</table>
2.7 acmsdi_cancel

TP Desktop Connector client programs call this service in a nonblocking or forced nonblocking environment to cancel a currently active ACMS task. Use the acmsdi_cancel service only if you invoke a task using nonblocking services. Do not use the acmsdi_cancel service from a presentation procedure or from an asynchronous completion routine.

Format

```
acmsdi_cancel (submitter id,
               call_id,
               [cancel_reason],
               reserved,
               [completion_status],
               completion_routine, 1
               [call_context])
```

Parameters

**submitter_id**
Type: ACMSDI_SUBMITTER_ID
Access: read
Mechanism: by reference
The submitter_id for the session associated with the task that is being canceled.

**call_id**
Type: ACMSDI_CALL_ID
Access: read
Mechanism: by reference
The call_id for the task that is being canceled, which was passed back on the acmsdi_call_task service.

**cancel_reason**
Type: long int
Access: read
Mechanism: by value
Optional parameter containing the status value of the reason code for the cancel request. This value is passed to the Application Execution Controller (EXC) by the TP Desktop Connector gateway. The default is ACMSDI_CALL_CANCELED, “the task was canceled by the task submitter”.

---

1 For nonblocking only, see Section 2.3.
acmsdi_cancel

reserved
Type:
Access:
Mechanism:
This parameter is reserved for future use. Specify as NULL.

completion_status
Type: long int
Access: write
Mechanism: by reference
Optional parameter to contain the final completion status of the service. The completion_status is set to ACMSDI_PENDING when the service starts successfully. When the service is successful, completion_status is set to 0.

completionRoutine
Type: function address\(^1\)
Access: read
Mechanism: by value
Address of a function to be called when the service completes. The completion_routine is called by the ACMSDI_DISPATCH_MESSAGE service when the appropriate reply is received from the TP Desktop Connector gateway on the OpenVMS system.

call_context
Type: void*
Access: read
Mechanism: by value
Optional parameter that is passed to the completion_routine to identify the call. You can use this to supply application-specific context for the call that is being canceled.

Return Status
The status values returned by the acmsdi_cancel service are listed in Table 2–3.

\(^1\) For nonblocking only, see Section 2.3. Not applicable to forced nonblocking and will cause an error if supplied.
### acmsdi_cancel

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_CANCELACTV</td>
<td>Cancel already in progress.</td>
</tr>
<tr>
<td>ACMSDI_EXCHACTV</td>
<td>Service cannot be called from presentation procedure.</td>
</tr>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>Insufficient parameters.</td>
</tr>
<tr>
<td>ACMSDI_INTERNAL</td>
<td>Internal ACMS error.</td>
</tr>
<tr>
<td>ACMSDI_INVCALLID</td>
<td>Invalid or obsolete call identification.</td>
</tr>
<tr>
<td>ACMSDI_INVSUBID</td>
<td>Invalid or obsolete submitter identification.</td>
</tr>
<tr>
<td>ACMSDI_NOMEMORY</td>
<td>Insufficient memory to complete requests.</td>
</tr>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_PENDING</td>
<td>Successful operation pending nonblocking completion. The final status is in the completion status parameter.</td>
</tr>
<tr>
<td>ACMSDISIGNINACTV</td>
<td>Request is invalid while the sign-in is active.</td>
</tr>
<tr>
<td>ACMSDISIGNOUTACTV</td>
<td>Request is invalid while sign-out is active.</td>
</tr>
<tr>
<td>ACMSDI_SRVDEAD</td>
<td>Node name is invalid, or the TP Desktop Connector gateway is not running on the specified node.</td>
</tr>
</tbody>
</table>
2.8 acmsdi_complete_pp

TP Desktop Connector client programs call this nonblocking service to complete exchange step processing for a submitter. An application must call this service to complete an outstanding presentation procedure request from the TP Desktop Connector gateway in a nonblocking or forced nonblocking environment (see HP TP Desktop Connector for ACMS Client Application Programming Guide).

Format

```c
acmsdi_complete_pp (call_id,
                     pp_status)
```

Parameters

call_id
Type: ACMSDI_CALL_ID
Access: read
Mechanism: by reference
The call_id parameter is passed back on the acmsdi_call_task service.

pp_status
Type: long int
Access: read
Mechanism: by value
The completion status of the presentation procedure. The pp_status parameter is returned to the ACMS task as the completion status for the current exchange step. A valid OpenVMS status value is returned to the task.

Return Status

The status values returned by the acmsdi_complete_pp service are listed in Table 2–4.
<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>Insufficient parameters.</td>
</tr>
<tr>
<td>ACMSDI_INTERNAL</td>
<td>Internal TP Desktop Connector error.</td>
</tr>
<tr>
<td>ACMSDI_INVCALLID</td>
<td>Invalid or obsolete call identification.</td>
</tr>
<tr>
<td>ACMSDI_INVSUBID</td>
<td>Invalid or obsolete submitter identification.</td>
</tr>
<tr>
<td>ACMSDI_NOMEMORY</td>
<td>Insufficient memory to complete requests.</td>
</tr>
<tr>
<td>ACMSDI_NOPPACTV</td>
<td>No presentation procedure active for this call.</td>
</tr>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_SRVDEAD</td>
<td>Node name is invalid, or the TP Desktop Connector gateway is not running on the specified node, or the network link terminated.</td>
</tr>
</tbody>
</table>
2.9 acmsdi_dispatch_message

TP Desktop Connector client programs call this nonblocking service to check for and process messages sent from a TP Desktop Connector gateway to an active submitter in the desktop application. The application must periodically call this service in a nonblocking environment to check for completion of outstanding acmsdi_sign_in, acmsdi_call_task, and acmsdi_sign_out requests. If no TP Desktop Connector messages are received, the service returns immediately. If a TP Desktop Connector message is received, the service calls the appropriate completion routine or presentation procedure and then returns (see HP TP Desktop Connector for ACMS Client Application Programming Guide).

Note that this call is not used in the forced nonblocking environment. See Section 4.13.

Format

acmsdi_dispatch_message ()

Parameters

No parameters are specified.

Return Status

The status values returned by the acmsdi_dispatch_message service are listed in Table 2–5.

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_INTERNAL</td>
<td>Internal TP Desktop Connector error.</td>
</tr>
<tr>
<td>ACMSDI_NOMEMORY</td>
<td>Insufficient memory.</td>
</tr>
</tbody>
</table>

(continued on next page)
### Table 2–5 (Cont.) acmsdi_dispatch_message Return Status Values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_SRVDEAD</td>
<td>Node name is invalid, or TP Desktop Connector gateway is not running on the specified node, or the network link terminated.</td>
</tr>
</tbody>
</table>
2.10 acmsdi_return_pointer

TP Desktop Connector client programs written in Visual Basic use the ACMSDI_RETURN_POINTER service to create the workspace array for the ACMSDI_CALL_TASK routine. When passing a workspace, the ACMSDI_CALL_TASK service expects a data structure with the size and address of each workspace. The ACMSDI_RETURN_POINTER service assigns the address of a workspace argument to the contents of a pointer address argument. This service may be used in the forced nonblocking environment to obtain reference pointers to structures such as call_id.

Format

acmsdi_return_pointer (structure)

Parameters

structure
Type: long int
Access: read
Mechanism: by reference
The workspace or other structure for which a pointer is to be obtained.

Return Status

The return status value for acmsdi_return_pointer is the address of the structure passed as the parameter in the call.
acmsdi_sign_in

2.11 acmsdi_sign_in

TP Desktop Connector client programs call this service to sign a user in to a ACMS system.

Format

```
acmsdi_sign_in (submitter_node,
    username,
    password,
    [options],
    submitter_id,
    [completion_status],
    [completion_routine], ¹
    [call_context])
```

Parameters

- **submitter_node**
  - Type: `char *`
  - Access: `read`
  - Mechanism: `by reference`
  - The node name of the ACMS system where the user is to be signed in. Maximum length is 80.

- **username**
  - Type: `char *`
  - Access: `read`
  - Mechanism: `by reference`
  - The name of the OpenVMS account of the user to be signed in. Maximum length is 80.

- **password**
  - Type: `char *`
  - Access: `read`
  - Mechanism: `by reference`
  - The password of the user to be signed in. Maximum length is 80.

¹ For nonblocking only, see Section 2.3.
options
Type: ACMSDI_OPTION array
Access: read
Mechanism: by reference
Union containing multiple structures and an option variable, the value of which defines the type of option being selected (see Section 2.5.1).

submitter_id
Type: ACMSDI_SUBMITTER_ID
Access: write
Mechanism: by reference
A structure into which the acmsdi_sign_in service writes a newly created submitter identification. Other services use the submitter identification as a handle to identify an active submitter. The ACMSDI_SUBMITTER_ID structure is defined in the ACMSDI.H include file.

completion_status
Type: int
Access: write
Mechanism: by reference
The final status of the service. In the blocking environment, the completion_status parameter is set to zero when the service starts successfully.

When the service completes, completion_status contains the final status. See Table 2–6 for a list of return status values.

completion_routine
Type: function address
Access: read
Mechanism: by value
Address of a function to be called when the nonblocking service completes. The completion routine is called by the acmsdi_dispatch_message service when the reply is received from the TP Desktop Connector gateway.

call_context
Type: void *
Access: read
Mechanism: by value
Optional parameter passed to presentation procedures and completion routines to identify the call. Use this parameter to supply application-specific context for the call.

---

For nonblocking only, see Section 2.3. Not applicable to forced nonblocking and will cause an error if supplied.
acmsdi_sign_in

Return Status

The status values returned by the acmsdi_sign_in service are listed in Table 2–6.

Table 2–6  acmsdi_sign_in Return Status Values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_CALLACTV</td>
<td>Call is active.</td>
</tr>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>Insufficient parameters.</td>
</tr>
<tr>
<td>ACMSDI_INTERNAL</td>
<td>Internal TP Desktop Connector error.</td>
</tr>
<tr>
<td>ACMSDI_INVLOGIN</td>
<td>Invalid login attempt.</td>
</tr>
<tr>
<td>ACMSDI_INVOPTION</td>
<td>Invalid item in options list.</td>
</tr>
<tr>
<td>ACMSDI_INVPROTOCOL</td>
<td>Mismatch in versions of TP Desktop Connector client and gateway software.</td>
</tr>
<tr>
<td>ACMSDI_MIXEDMODE</td>
<td>All calls on a connection must be either blocking or nonblocking.</td>
</tr>
<tr>
<td>ACMSDI_NOACMS</td>
<td>ACMS system not available.</td>
</tr>
<tr>
<td>ACMSDI_NOCOMPRESS</td>
<td>Gateway does not allow compression.</td>
</tr>
<tr>
<td>ACMSDI_NOMEMORY</td>
<td>Insufficient memory to complete requests.</td>
</tr>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_PENDING</td>
<td>Successful operation pending nonblocking completion. The final status is in the completion_status parameter.</td>
</tr>
<tr>
<td>ACMSDI_PWDEXPIRED</td>
<td>Password has expired.</td>
</tr>
<tr>
<td>ACMSDI_PWDEXPIRING</td>
<td>Password expiring in the number of hours specified in options array.</td>
</tr>
<tr>
<td>ACMSDI_SIGNINACTV</td>
<td>Sign-in active.</td>
</tr>
<tr>
<td>ACMSDI_SIGNOUTACTV</td>
<td>Sign-out active.</td>
</tr>
<tr>
<td>ACMSDI_SRVDEAD</td>
<td>Node name is invalid, or TP Desktop Connector gateway is not running on the specified node, or the network link terminated.</td>
</tr>
</tbody>
</table>
acmsdi_sign_out

2.12 acmsdi_sign_out

TP Desktop Connector client programs call this service to terminate an active session with a ACMS system. To insure that all network links are properly shut down, the desktop client program calls the acmsdi_sign_out service before terminating.

Format

```plaintext
acmsdi_sign_out (submitter_id,
    [completion_status],
    [completion_routine], ¹
    [call_context])
```

Parameters

submitter_id
Type: ACMSDI_SUBMITTER_ID
Access: read
Mechanism: by reference
The submitter identification returned by the acmsdi_sign_in service.

completion_status
Type: int
Access: write
Mechanism: by reference
The final status of the service. In the blocking environment, the completion_status parameter is set to zero when the service starts successfully.

When the service completes, completion_status contains the final status. See Table 2–7 for a list of the return status values.

completion_routine
Type: function address¹
Access: read
Mechanism: by value
Address of a function to be called when the nonblocking service completes. The completion routine is called by the acmsdi_dispatch_message service when the reply is received from the TP Desktop Connector gateway.

¹ For nonblocking only, see Section 2.3.
acmsdi_sign_out

call_context
Type: void *
Access: read
Mechanism: by value
Optional parameter passed to presentation procedures and completion routines to identify the call. Use this parameter to supply application-specific context for the call.

Return Status

The status values returned by the acmsdi_sign_out service are listed in Table 2–7.

Table 2–7 acmsdi_sign_out Return Status Values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_CALLACTV</td>
<td>Request is invalid while task call is active.</td>
</tr>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>Insufficient parameters.</td>
</tr>
<tr>
<td>ACMSDI_INTERNAL</td>
<td>Internal TP Desktop Connector error.</td>
</tr>
<tr>
<td>ACMSDI_INVSUBID</td>
<td>Invalid or obsolete submitter identification.</td>
</tr>
<tr>
<td>ACMSDI_MIXEDMODE</td>
<td>All calls on a connection must be either blocking or nonblocking.</td>
</tr>
<tr>
<td>ACMSDI_NOMEMORY</td>
<td>Insufficient memory to complete requests.</td>
</tr>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_PENDING</td>
<td>Successful operation pending nonblocking completion. The final status is in the completion_status parameter.</td>
</tr>
<tr>
<td>ACMSDI_SIGNINACTV</td>
<td>Request is invalid while sign-in is active.</td>
</tr>
</tbody>
</table>

(continued on next page)
### acmsdi_sign_out

**Table 2–7 (Cont.) acmsdi_sign_out Return Status Values**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_SIGNOUTACTV</td>
<td>Request is invalid while sign-out is active.</td>
</tr>
<tr>
<td>ACMSDI_SRVDEAD</td>
<td>Node name is invalid, or TP Desktop Connector gateway is not running on the specified node, or the network link terminated.</td>
</tr>
</tbody>
</table>
This chapter describes the interface between the TP Desktop Connector gateway and customer-written presentation procedures. It also describes the interfaces on portable clients for customer-written action routines to perform version checking.

3.1 Summary of Portable API Presentation Procedures

Presentation procedures are customer-written routines that the TP Desktop Connector gateway calls when an exchange step occurs in an ACMS task with either the FORM I/O or REQUEST I/O attribute. Table 3–1 summarizes the presentation procedures available in a nonblocking session. These are not applicable to a forced nonblocking session. For more information on presentation procedures, refer to HP TP Desktop Connector for ACMS Client Application Programming Guide.

Table 3–1 Summary of Portable API Presentation Procedures

<table>
<thead>
<tr>
<th>Customer-Supplied Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acmsdi_disable</td>
<td>Services a HP DECforms disable request, which disables a form.</td>
</tr>
<tr>
<td>acmsdi_enable</td>
<td>Services a HP DECforms enable request, which enables a form.</td>
</tr>
<tr>
<td>acmsdi_read_msg</td>
<td>Services a TDMS Read exchange, which displays the prompt, if any, sent from the ACMS task, and acquires the text from the form's message field.</td>
</tr>
<tr>
<td>acmsdi_receive</td>
<td>Services a HP DECforms receive request, which sends data from the form to the application program.</td>
</tr>
</tbody>
</table>

(continued on next page)
Table 3–1 (Cont.) Summary of Portable API Presentation Procedures

<table>
<thead>
<tr>
<th>Customer-Supplied Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acmsdi_request</td>
<td>Services a TDMS Request exchange, which displays a form and transfers data between a form and the application program.</td>
</tr>
<tr>
<td>acmsdi_send</td>
<td>Services a HP DECforms send request, which sends data from the application program to the form.</td>
</tr>
<tr>
<td>acmsdi_transceive</td>
<td>Services a HP DECforms transceive request, which combines the actions of a send and a receive.</td>
</tr>
<tr>
<td>acmsdi_write_msg</td>
<td>Services a TDMS Write exchange, which displays the text sent from the form's message field or the ACMS task.</td>
</tr>
</tbody>
</table>

See *HP TP Desktop Connector for ACMS Client Application Programming Guide* for a description of sample client presentation procedures.

### 3.1.1 Return Status Values Expected from Presentation Procedures

The presentation procedure routines must return a long integer containing any valid OpenVMS status value, including HP DECforms, TDMS, and application-defined values. Return status values for nonblocking presentation procedures are returned using the acmsdi_complete_pp routine. The status value is passed to the ACMS Application Execution Controller (EXC) as the completion status for the exchange step. The EXC attempts to interpret the value as a standard OpenVMS status value. Unless the task definition for the exchange step specifies CONTINUE ON FAILURE, the EXC cancels the task for an error status returned.

The TP Desktop Connector kit provides include files that specify the return status values for HP DECforms and TDMS: FORMS.H and TDMS.H. If the return status values change, you can regenerate the include files with the command procedures, MAKE_FORMS_H.COM and MAKE_TDMS_H.COM, in the ACMSDI$EXAMPLES directory.

To handle errors, specify the exception-handler syntax in the task definition. To have a single ACMS application support both HP DECforms terminals and graphical desktop systems, code the task definition to check for a HP DECforms error status value.
3.1.2 ACMSDI_FORM_RECORD Structure and Macro Call

Defined in the ACMSDI.H file, the ACMSDI_FORM_RECORD type declares form records and shadow records passed to and from presentation procedures. The code in Example 3–1 defines the ACMSDI_FORM_RECORD type and a macro ACMSDI_INIT_FORM_RECORD to initialize the form record structure.

Example 3–1 Form Record Definition and Initialization Macro

```c
typedef struct {
    int data_length;    /**< length of data record **/
    void *data_record;  /**< pointer to data record **/
    int shadow_length;  /**< length of shadow record **/
    void *shadow_record;/**< pointer to shadow record **/
} ACMSDI_FORM_RECORD;

#define ACMSDI_INIT_FORM_RECORD (record, data, shadow) {
    record.data_length = sizeof(data);
    record.data_record = &data;
    record.shadow_length = sizeof(shadow);
    record.shadow_record = &shadow;
}
```

3.1.3 Prototypes and Code for Presentation Procedures and Version Routines

The ACMSDI.H file contains function prototypes for the presentation procedures and action routines that your code supplies. The file PPSTUBS.C contains stub modules you can use for linking your application (see HP TP Desktop Connector for ACMS Client Application Programming Guide).

3.2 Parameter Memory Allocation

The caller of a TP Desktop Connector service or presentation procedure is responsible for allocating the memory for the parameters of that routine. For calls to the TP Desktop Connector client services, the desktop client program must allocate the memory for all parameters passed in, for example, submitterr_id and call_context. For the presentation procedures, the desktop client program can expect that TP Desktop Connector software allocates memory for all the parameters passed in and for all workspaces before it calls these procedures.
3.3 Blocking and Nonblocking Usage

Like the portable TP Desktop Connector client services, presentation procedures can be either blocking or nonblocking. If the desktop client program supplies the `completion_routine` parameter in the `acmsdi_call_task` call, the service behaves in the nonblocking environment (see Section 2.3). In a nonblocking environment, presentation procedures must behave in a way consistent with nonblocking services.

3.3.1 Presentation Procedures in a Nonblocking Environment

When nonblocking services are in use, presentation procedures are written in two parts:

- The first part handles the generic presentation procedure and dispatches to the application-specific presentation procedure to handle interaction with the user.
- The second part uses the `acmsdi_complete_pp` service to indicate that exchange step processing is completed.

The TP Desktop Connector client services return exchange step data and status to the TP Desktop Connector gateway when the desktop client program calls the `acmsdi_complete_pp` service.

3.3.2 Nonblocking and Blocking Restriction

All calls using the same desktop client program and TP Desktop Connector gateway connection must be either blocking, nonblocking, or forced nonblocking. These types of service calls cannot be mixed for a client/server pair. If a desktop client program connects to two different TP Desktop Connector gateways, it can mix service call types, using blocking calls to interact with one gateway and nonblocking calls to interact with the other gateway.
TP Desktop Connector services call this procedure for each active forms session for a desktop submitter whenever the TP Desktop Connector client program calls acmsdi_sign_out to sign the submitter out of the ACMS system.

**Format**

```c
acmsdi_disable (forms_session, 
    call_id, 
    call_context)
```

**Parameters**

- **forms_session**
  - Type: ACMSDI_FORMS_SESSION_ID
  - Access: read
  - Mechanism: by reference
  - An identification that associates the session with the form specified in the acmsdi_enable request (see Section 3.5).

- **call_id**
  - Type: ACMSDI_CALL_ID
  - Access: read
  - Mechanism: by reference
  - The call identification used to complete the disable call when using nonblocking services. See the description of acmsdi_complete_pp (Section 2.8).

- **call_context**
  - Type: void *
  - Access: read
  - Mechanism: by value
  - Application-specific context for the call. This is the same context that was passed by the application to the acmsdi_sign_out() call.

**Return Status**

The status values returned by the acmsdi_disable procedure are described in Section 3.1.1.
3.5 acmsdi_enable

TP Desktop Connector client services call this presentation procedure whenever a HP DECforms ENABLE request is received from the TP Desktop Connector gateway on the OpenVMS system.

Format

\[
\text{acmsdi}_\text{enable} \ (\text{submitter\_id}, \\
\quad \text{forms\_session}, \\
\quad \text{file\_specification}, \\
\quad \text{form\_specification}, \\
\quad \text{forms\_print\_file}, \\
\quad \text{forms\_language}, \\
\quad \text{call\_id}, \\
\quad \text{call\_context})
\]

Parameters

\textbf{submitter\_id}
Type: \textit{ACMSDI\_SUBMITTER\_ID}
Access: \textbf{read}
Mechanism: \textbf{by reference}
The value returned by the acmsdi\_sign\_in service (see Section 2.11).

\textbf{forms\_session}
Type: \textit{ACMSDI\_FORMS\_SESSION\_ID}
Access: \textbf{write}
Mechanism: \textbf{by reference}
An identification that associates the session with the submitter identification. This is a write parameter that customer-supplied presentation procedures can fill. Presentation procedures (acmsdi\_send, acmsdi\_receive, acmsdi\_transceive, and acmsdi\_disable) can use the \textit{forms\_session} parameter to associate the session with the form specified in the enable request. The TP Desktop Connector run-time system passes this parameter to subsequent requests to specify which form to use.
file_specification
Type: char *
Access: read
Mechanism: by reference
The form file specification from the ACMS task group definition. Refer to *HP TP Desktop Connector for ACMS Client Application Programming Guide* for guidelines on specifying the form file specification.

form_specification
Type: char *
Access: read
Mechanism: by reference
The form name specified in the exchange step in the ACMS task definition. Refer to *HP TP Desktop Connector for ACMS Client Application Programming Guide* for guidelines on specifying the form name.

forms_print_file
Type: char *
Access: read
Mechanism: by reference
The HP DECforms specification for the user in ACMSUDF.DAT.

forms_language
Type: char *
Access: read
Mechanism: by reference
The HP DECforms specification for the user in ACMSUDF.DAT.

call_id
Type: ACMSDI_CALL_ID
Access: read
Mechanism: by reference
The call identification returned by the acmsdi_call_task service.

call_context
Type: void *
Access: read
Mechanism: by value
Application-specific context for the call. This is the same context that was passed by the application to the acmsdi_call_task() call.
acmsdi_enable

Return Status

The status values returned by the acmsdi_enable procedure are described in Section 3.1.1.
3.6 acmsdi_read_msg

TP Desktop Connector client services call this presentation procedure when a TDMS Read exchange is received from the TP Desktop Connector gateway on the host OpenVMS system. Its function is to display the prompt (if any) sent from the ACMS task and then to acquire the text from the form's Message Field to be returned to ACMS.

Format

\[
\text{acmsdi\_read\_msg} \ (\text{submitter\_id}, \\
\text{msg\_text}, \\
\text{prompt\_text}, \\
\text{call\_id}, \\
\text{call\_context})
\]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Access</th>
<th>Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>submitter_id</td>
<td>ACMSDI_SUBMITTER_ID</td>
<td>read</td>
<td>by reference</td>
</tr>
<tr>
<td>msg_text</td>
<td>char</td>
<td>write</td>
<td>by reference</td>
</tr>
<tr>
<td>prompt_text</td>
<td>char</td>
<td>read</td>
<td>by reference</td>
</tr>
</tbody>
</table>

- **submitter_id**: The value returned by the acmsdi_sign_in service.
- **msg\_text**: A buffer into which the presentation procedure will write the text from the form's Message Field to be returned to the ACMS task. This is a C-style null-terminated string with a maximum length of 132 plus one for the null terminator.
- **prompt\_text**: Text to be displayed as a prompt to the terminal operator. This is a C-style null-terminated string with a maximum length of 132 plus one for the null terminator. There may be no prompt text to display in which case the length will be 0; that is, the null terminator will be in the first position.
acmsdi_read_msg

call_id
Type: ACMSDI_CALL_ID
Access: read
Mechanism: by reference
The call identification returned by the acmsdi_call_task service which initiated
the ACMS task associated with this exchange.

call_context
Type: void *
Access: read
Mechanism: by value
Application-specific context for the call. This is the same context that was
passed by the application to the acmsdi_call_task service which initiated the
ACMS task associated with this exchange.

Return Status
This function returns a ps32, defined in ACMSDI.H to be equivalent to a
signed 32-bit integer. The value must be a valid TDMS status code. Valid
TDMS statuses are defined in TDMS.H.
3.7 acmsdi_receive

The TP Desktop Connector client services call this presentation procedure whenever a HP DECforms RECEIVE request is received from the TP Desktop Connector gateway on the OpenVMS system.

Format

acmsdi_receive (forms_session,
               receive_record_identifier,
               receive_record_count,
               receive_control_text,
               receive_control_text_count,
               send_control_text,
               send_control_text_count,
               timeout,
               call_id,
               call_context,
               receive_record)

Parameters

forms_session
Type: ACMSDI_FORMS_SESSION_ID
Access: read
Mechanism: by reference
An identification to associate the session with the form specified in the acmsdi_enable request (see Section 3.5).

receive_record_identifier
Type: char *
Access: read
Mechanism: by reference
The form record name or record list name specified in the RECEIVE request in the ACMS task. Refer to HP TP Desktop Connector for ACMS Client Application Programming Guide for guidelines on specifying the form name.

receive_record_count
Type: long int
Access: read
Mechanism: by value
The number of receive record items sent from the ACMS task.
acmsdi_receive

**receive_control_text**
Type: `char *`  
Access: `write`  
Mechanism: `by reference`  
A 25-character string that the customer-supplied request can use to return receive control text.

**receive_control_text_count**
Type: `long int`  
Access: `write`  
Mechanism: `by reference`  
The number of receive control text items that the customer-supplied request returns.

**send_control_text**
Type: `char *`  
Access: `read`  
Mechanism: `by reference`  
Send control text sent from the ACMS task.

**send_control_text_count**
Type: `long int`  
Access: `read`  
Mechanism: `by value`  
The number of send control text items sent from the ACMS task.

**timeout**
Type: `short int`  
Access: `read`  
Mechanism: `by value`  
A timeout value for user input processing sent from the ACMS task.

**call_id**
Type: `ACMSDI_CALL_ID`  
Access: `read`  
Mechanism: `by reference`  
The call identification returned by the acmsdi_call_task service.

**call_context**
Type: `void *`  
Access: `read`  
Mechanism: `by value`  
Application-specific context for the call. This is the same context that was passed by the application to the acmsdi_call_task() call.
acmsdi_receive

receive_record
Type: ACMSDI_FORM_RECORD array  
Access: write  
Mechanism: by reference  
An array of ACMSDI_FORM_RECORD structures pointing to buffers that store application data and shadow records from the request (see Section 3.1.2).

Return Status

The status values returned by the acmsdi_receive procedure are described in Section 3.1.1.
acmsdi_request

3.8 acmsdi_request

TP Desktop Connector client services call this presentation procedure whenever a TDMS Request exchange is received from the TP Desktop Connector gateway on the OpenVMS system.

Format

acmsdi_request (submitter_id,
request_name,
workspace_count,
workspaces,
call_id,
call_context)

Parameters

submitter_id
Type: ACMSDI_SUBMITTER_ID
Access: read
Mechanism: by reference
The value returned by the acmsdi_sign_in service (see Section 2.11).

request_name
Type: char *
Access: read
Mechanism: by reference
The name of the TDMS request specified in the ACMS task.

workspace_count
Type: long int
Access: read
Mechanism: by value
The number of workspaces sent from the ACMS task.
workspaces
Type: ACMSDI_WORKSPACE array
Access: read/write
Mechanism: by reference
The workspace data sent from the ACMS task. One or more optional workspace arguments can be sent from the task (see Section 2.4).

call_id
Type: ACMSDI_CALL_ID
Access: read
Mechanism: by reference
The call identification returned by the acmsdi_call_task service.

call_context
Type: void *
Access: read
Mechanism: by value
Application-specific context for the call. This is the same context that was passed by the application to the acmsdi_call_task() call.

Return Status
The status values returned by the acmsdi_request procedure are described in Section 3.1.1.
3.9 acmsdi_send

TP Desktop Connector client services call this presentation procedure whenever a HP DECforms SEND request is received from the TP Desktop Connector gateway on the OpenVMS system.

Format

```
acmsdi_send (forms_session,
             send_record_identifier,
             send_record_count,
             receive_control_text,
             receive_control_text_count,
             send_control_text,
             send_control_text_count,
             timeout,
             call_id,
             call_context,
             send_record)
```

Parameters

- **forms_session**
  - Type: `ACMSDI_FORMS_SESSION_ID`
  - Access: read
  - Mechanism: by reference
  - An identification that associates the session with the form specified in the acmsdi_enable request (see Section 3.5).

- **send_record_identifier**
  - Type: `char *`
  - Access: read
  - Mechanism: by reference
  - The form record name or record list name specified in the SEND request in the ACMS task. Refer to *HP TP Desktop Connector for ACMS Client Application Programming Guide* for guidelines on specifying the form name.

- **send_record_count**
  - Type: `long int`
  - Access: read
  - Mechanism: by value
  - The number of send record items sent from the ACMS task.
acmsdi_send

receive_control_text
Type: char *
Access: write
Mechanism: by reference
A 25-character string that the customer-supplied request can use to return receive control text.

receive_control_text_count
Type: long int
Access: write
Mechanism: by reference
The number of receive control text items that the customer-supplied request returns.

send_control_text
Type: char *
Access: read
Mechanism: by reference
Send control text sent from the ACMS task.

send_control_text_count
Type: long int
Access: read
Mechanism: by value
The number of send control text items sent from the ACMS task.

timeout
Type: short int
Access: read
Mechanism: by value
A timeout value for user input processing, sent from the ACMS task.

call_id
Type: ACMSDI_CALL_ID
Access: read
Mechanism: by reference
The call identification returned by the acmsdi_call_task service.

call_context
Type: void *
Access: read
Mechanism: by value
Application-specific context for the call. This is the same context that was passed by the application to the acmsdi_call_task() call.
acmsdi_send

send_record
Type: ACMSDI_FORM_RECORD array
Access: read
Mechanism: by reference
An array of ACMSDI_FORM_RECORD structures pointing to buffers containing application data and shadow records sent from the ACMS task (see Section 3.1.2).

Return Status
The status values returned by the acmsdi_send procedure are described in Section 3.1.1.
3.10 acmsdi_transceive

TP Desktop Connector client services call this presentation procedure whenever a HP DECforms TRANSCEIVE request is received from the TP Desktop Connector gateway on the OpenVMS system.

Format

acmsdi_transceive (forms_session,
                send_record_identifier,
                send_record_count,
                receive_record_identifier,
                receive_record_count,
                receive_control_text,
                receive_control_text_count,
                send_control_text,
                send_control_text_count,
                timeout,
                call_id,
                call_context,
                send_record,
                receive_record)

Parameters

forms_session
Type: ACMSDI_FORMS_SESSION_ID
Access: read
Mechanism: by reference
An identification that associates the session with the form specified in the acmsdi_enable request (see Section 3.5).

send_record_identifier
Type: char *
Access: read
Mechanism: by reference
The form record name or record list name specified in the SEND request in the ACMS task. Refer to HP TP Desktop Connector for ACMS Client Application Programming Guide for guidelines on specifying the form name.
acmsdi_transceive

**send_record_count**
Type: long int  
Access: read  
Mechanism: by value  
The number of send record items sent from the ACMS task.

**receive_record_identifier**
Type: char *  
Access: read  
Mechanism: by reference  
The form record name or record list name specified in the RECEIVE request in the ACMS task. Refer to HP TP Desktop Connector for ACMS Client Application Programming Guide for guidelines on specifying the form name.

**receive_record_count**
Type: long int  
Access: read  
Mechanism: by value  
The number of receive record items sent from the ACMS task.

**receive_control_text**
Type: char *  
Access: write  
Mechanism: by reference  
A 25-character string that the customer-supplied request can use to return receive control text.

**receive_control_text_count**
Type: long int  
Access: write  
Mechanism: by reference  
The number of receive control text items that the customer-supplied request returns.

**send_control_text**
Type: char *  
Access: read  
Mechanism: by reference  
Send control text sent from the ACMS task.
send_control_text_count
Type: long int
Access: read
Mechanism: by value
The number of send control text items sent from the ACMS task.

timeout
Type: short int
Access: read
Mechanism: by value
A timeout value for user input processing, sent from the ACMS task.

call_id
Type: ACMSDI_CALL_ID
Access: read
Mechanism: by reference
The call identification returned by the acmsdi_call_task service.

call_context
Type: void *
Access: read
Mechanism: by value
Application-specific context for the call. This is the same context that was passed by the application to the acmsdi_call_task() call.

send_record
Type: ACMSDI_FORM_RECORD array
Access: read
Mechanism: by reference
An array of ACMSDI_FORM_RECORD structures pointing to buffers containing application data and shadow records sent from the ACMS task (see Section 3.1.2).

receive_record
Type: ACMSDI_FORM_RECORD array
Access: write
Mechanism: by reference
An array of ACMSDI_FORM_RECORD structures pointing to buffers to receive application data and shadow records from the request (see Section 3.1.2).
acmsdi_transceive

Return Status

The status values returned by the acmsdi_transceive procedure are described in Section 3.1.1.
3.11 acmsdi_write_msg

TP Desktop Connector client services call this presentation procedure when a TDMS Write exchange is received from the TP Desktop Connector gateway on the host OpenVMS system. Its function is to display the message text sent from the ACMS task in the form’s Message Field.

Format

    acmsdi_write_msg (submitter_id,
        msg_text,
        call_id,
        call_context)

Parameters

submitter_id
Type: ACMSDI_SUBMITTER_ID
Access: read
Mechanism: by reference
The value returned by the acmsdi_sign_in service.

msg_text
Type: char
Access: read
Mechanism: by reference
Text to be displayed in the form’s Message Field. This is a C-style null-terminated string with a maximum length of 132 plus one for the null terminator.
acmsdi_write_msg

**call_id**
Type: ACMSDI_CALL_ID
Access: read
Mechanism: by reference
The call identification returned by the acmsdi_call_task service which initiated the ACMS task associated with this exchange.

**call_context**
Type: void *
Access: read
Mechanism: by value
Application-specific context for the call. This is the same context that was passed by the application to the acmsdi_call_task service which initiated the ACMS task associated with this exchange.

**Return Status**
This function returns a ps32, defined in ACMSDL.H to be equivalent to a signed 32-bit integer. The value must be a valid TDMS status code. Valid TDMS statuses are defined in TDMS.H.
3.12 Version-Checking Routines

The following sections describe the version-checking routines. Version checking is supported on systems using FORM I/O tasks (see HP TP Desktop Connector for ACMS Client Application Programming Guide).

3.12.1 acmsdi_check_version

TP Desktop Connector client services call this routine whenever they receive an ENABLE request from the TP Desktop Connector gateway. The action routine can check the version string passed from the acmsdi_get_version routine on the submitter node and notify the desktop user of any inconsistency.

You request version checking during a sign-in (see HP TP Desktop Connector for ACMS Client Application Programming Guide).

Format

acmsdi_check_version (form_file, version)

Parameters

form_file
Type: char *
Access: read
Mechanism: by reference
Specification of a form file or a request library from the ACMS task group definition.

version
Type: char *
Access: read
Mechanism: by reference
Twenty-four bytes containing the version number or date supplied by the acmsdi_get_version routine on the OpenVMS system.
Return Status

The TP Desktop Connector service checks the status value returned and expects a valid OpenVMS status. If a failure status is returned, the TP Desktop Connector run-time system terminates the ENABLE request.

If the version-checking routine determines that software is not synchronized, it does one of the following:

- Returns an OpenVMS failure status that cancels the ENABLE request.
- Sets a flag that causes the acmsdi_enable routine to terminate with a failure status.
The TP Desktop Connector gateway calls this routine on the OpenVMS system whenever it receives an ENABLE request from the EXC. The action routine can return a version string that is then passed to the desktop client program, allowing a version comparison at the desktop system.

This service can also be used in a forced nonblocking environment, see Section 4.3. On a Windows system, version checking occurs during enable processing.

**Format**

```c
acmsdi_get_version (form_file, version)
```

**Parameters**

- **form_file**
  - Type: `char *`
  - Access: `read`
  - Mechanism: `by reference`
  - Form file or request library specification from the ACMS task group definition.

- **version**
  - Type: `char *`
  - Access: `write`
  - Mechanism: `by reference`
  - Twenty-four bytes in which the routine writes the version number or date associated with the specified form file. The `version` parameter is passed to the desktop client program to be checked in the acmsdi_check_version routine.

**Return Status**

Always returns SUCCESS status.
This chapter describes the forced nonblocking interface between the TP Desktop Connector gateway and customer-written procedures.

4.1 Summary of Forced Nonblocking Procedures

Forced nonblocking client services extend the Portable API to support development tools that do not provide for callbacks, data pointers or consistent memory locations for data. (Such tools include Visual Basic and others.) You create a forced nonblocking session when you specify the ACMSDI_OPTION, ACMSDI_OPT_NONBLK, with the acmsdi_sign_in service and do not supply a completion address. In this session, all calls are nonblocking. Table 4–1 summarizes the forced nonblocking calls to the TP Desktop Connector API. For more information on forced nonblocking calls, refer to HP TP Desktop Connector for ACMS Client Application Programming Guide.

Table 4–1 Summary of Forced Nonblocking Procedures

<table>
<thead>
<tr>
<th>Customer-Supplied Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acmsdi_complete_call</td>
<td>Returns the completion status. Can also provide the ACMS status message and task argument workspaces.</td>
</tr>
<tr>
<td>acmsdi_bind_enable_args</td>
<td>Retrieves write-only arguments in an enable exchange step request.</td>
</tr>
<tr>
<td>acmsdi_bindEnable_args</td>
<td>Retrieves write-only arguments in an enable exchange step request.</td>
</tr>
<tr>
<td>acmsdi_bind_msg</td>
<td>Sends or acquires the message text in TDMS Read or Write exchanges, respectively, or acquires the prompt text of a TDMS Read exchange.</td>
</tr>
</tbody>
</table>

(continued on next page)
Table 4–1 (Cont.) Summary of Forced Nonblocking Procedures

<table>
<thead>
<tr>
<th>Customer-Supplied Procedure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acmsdi_bind_receive_recs</td>
<td>Services receive and transceive exchange steps, which send data from the desktop client to the TP Desktop Connector gateway.</td>
</tr>
<tr>
<td>acmsdi_bind_request_args</td>
<td>Provides the client application with the request name and identifies the set of workspaces in a TDMS request exchange step.</td>
</tr>
<tr>
<td>acmsdi_bind_request_wksp</td>
<td>Services a TDMS exchange step, which transfers data between a desktop client and the TP Desktop Connector gateway.</td>
</tr>
<tr>
<td>acmsdi_bind_send_args</td>
<td>Provides the client application with the send record identifier and identifies the records to be received in a send exchange step.</td>
</tr>
<tr>
<td>acmsdi_bind_send_recs</td>
<td>Services send and transceive exchange steps, which send data from the TP Desktop Connector gateway to the desktop client.</td>
</tr>
<tr>
<td>acmsdi_bind_session_id</td>
<td>Sends the forms session identifier to the TP Desktop gateway during an enable exchange step.</td>
</tr>
<tr>
<td>acmsdi_bind_transceive_args</td>
<td>Provides the client application with the send and receive record identifiers and identifies the records to be passed in a transceive exchange step.</td>
</tr>
<tr>
<td>acmsdi_poll</td>
<td>Returns the message type of a message from the back end and a pointer to the call context from the client application.</td>
</tr>
</tbody>
</table>

4.1.1 ACMSDI_FORM_RECORD_BIND Structure

Defined in the ACMSDI.H and ACMSDI.BAS files, the ACMSDI_FORM_RECORD type declares form records and shadow records transferred. The code in Example 4–1 defines the ACMSDI_FORM_RECORD_BIND type for the C language.
Example 4–1  Form Record Definition

typedef struct {
    unsigned int buffer_len; /**< length of caller’s record buffer **/
    unsigned int rec_len;   /**< actual length of the form record **/
    void  *data_record;     /**< pointer to data record **/
    unsigned int shadow_buffer_len; /**< length of callers shadow buffer **/
    unsigned int shadow_rec_len; /**< actual length of shadow record **/
    void  *shadow_record;    /**< pointer to shadow record /**/
} ACMSDI_FORM_RECORD_BIND;

4.1.2 ACMSDI_WORKSPACE_BIND Structure

Defined in the ACMSDI.H file, the ACMSDI_WORKSPACE_BIND type declares workspaces passed to tasks using the acmsdi_call_task service and workspaces passed from tasks to acmsdi_request presentation procedures.

The code in Example 4–2 defines the ACMSDI_WORKSPACE_BIND type structure.

Example 4–2  Workspace Structure Definition

typedef struct {
    unsigned int buffer_len; /**< length of caller’s buffer **/
    unsigned int wksp_len;  /**< actual length of the workspace /**/
    void  *data;            /**< pointer to workspace /**/
} ACMSDI_WORKSPACE_BIND;
The acmsdi_complete_call service is a required call that obtains completion arguments for acsmdi_call_task, acsmdi_sign_in, acsmdi_sign_out, and acsmdi_cancel services. When acsmdi_poll detects completion, acmsdi_complete_call can obtain the completion status for these services. The acmsdi_complete_call can also obtain the ACMS status message and task argument workspaces sent from the back end for the acsmdi_call_task service.

**Format**

```
acmsdi_complete_call (submitter_id,
 completion_status,
 [call_id],
 [status_message],
 [workspaces])
```

**Parameters**

- **submitter_id**
  - Type: `ACMSDI_SUBMITTER_ID`
  - Access: `read`
  - Mechanism: `by reference`
  - The submitter_id returned by the acmsdi_sign_in service.

- **completion_status**
  - Type: `int`
  - Access: `write`
  - Mechanism: `by reference`
  - The final status of the TP Desktop Connector client service. When the service completes, the `completion_status` parameter contains the final status. For the list of return status values, see Table 4–2.

  When a task is canceled, the TP Desktop Connector gateway reports a specific error, where possible. If the gateway cannot convert a ACMS error to a specific TP Desktop Connector status, it returns ACMSDI_TASK_FAILED to the desktop client program.

- **call_id**
  - Type: `ACMSDI_CALL_ID` *
  - Access: `read`
  - Mechanism: `by reference`
A structure defined in the ACMSDI.H include file into which the acmsdi_call_task service writes a newly created call identification, a handle used by the TP Desktop Connector client services to identify an active call for a submitter. This parameter is required when completing an acmsdi_call_task service.

**status_message**
Type: char *
Access: write
Mechanism: by reference
A buffer to receive the message text associated with the task completion status. The message text returned is either the text associated with a TP Desktop Connector error or the message text returned from a ACMS application. Required length is 80. You use this parameter only for the acmsdi_call_task service completion.

---
**Caution**
If the full space is not allocated, the TP Desktop Connector client services write past the end of the allocated string and can cause the application to fail. Ensure that the desktop client program allocates the required length of space.

---

**workspaces**
Type: ACMSDI_WORKSPACE or ACMSDI_WORKSPACE_OPT array
Access: write
Mechanism: by reference
One or more optional workspaces passed to the application from the back end. You need to typecast your array to void *. The workspaces must be specified in the same order as they are declared in the task definition, and must match the number specified in the workspace_count parameter. If you use the ACMSDI_WORKSPACE_OPT type, you must set the call_options parameter to allow unidirectional workspaces.

Because buffers may have been relocated by memory management, workspace pointers in the structures must be renewed using the acmsdi_return_pointer call just prior to issuing acmsdi_complete_call.

You use this parameter only for the acmsdi_call_task service completion.
The status values returned by the acmsdi_complete_call procedure are described in Table 4–2.

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_EXCHACTV</td>
<td>Request is invalid while exchange step processing is active.</td>
</tr>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>Insufficient parameters.</td>
</tr>
<tr>
<td>ACMSDI_INTERNAL</td>
<td>Internal TP Desktop Connector error.</td>
</tr>
<tr>
<td>ACMSDI_INVCALLID</td>
<td>Invalid or obsolete call identification.</td>
</tr>
<tr>
<td>ACMSDI_INVSUBID</td>
<td>Invalid or obsolete submitter identification.</td>
</tr>
<tr>
<td>ACMSDI_MIXEDMODE</td>
<td>Not a forced nonblocking session.</td>
</tr>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_WRONG_STATE</td>
<td>Session is in the wrong state for this call.</td>
</tr>
</tbody>
</table>
4.3  acmsdi_bind_enable_args

The client application can call this service whenever acmsdi_poll returns ACMSDI_ENABLE_EXCH from the TP Desktop Connector gateway on the OpenVMS system. This service retrieves the write-only arguments passed from the TP Desktop Connector client services. This is an optional call.

Format

acmsdi_bind_enable_args (submitter_id,
                      file_specification,
                      form_specification,
                      form_version,
                      forms_print_file,
                      forms_language,
                      call_id)

Parameters

submitter_id
Type:  ACMSDI_SUBMITTER_ID
Access:  read
Mechanism:  by reference
The value returned by the acmsdi_sign_in service (see Section 2.11).

file_specification
Type:  char *
Access:  write
Mechanism:  by reference
The form file specification from the ACMS task group definition. Refer to HP TP Desktop Connector for ACMS Client Application Programming Guide for guidelines on specifying the form file specification.

form_specification
Type:  char *
Access:  write
Mechanism:  by reference
The form name specified in the exchange step in the ACMS task definition. Refer to HP TP Desktop Connector for ACMS Client Application Programming Guide for guidelines on specifying the form name.
acmsdi_bind_enable_args

form_version
Type: char *
Access: write
Mechanism: by reference
Twenty-four bytes containing the version number or date supplied by the
acmsdi_get_version routine on the OpenVMS system. The argument provides
for version checking by the client application. (The acmsdi_check_version is
not available in a forced nonblocking session.)

forms_print_file
Type: char *
Access: write
Mechanism: by reference
The HP DECforms specification for the user in ACMSUDF.DAT.

forms_language
Type: char *
Access: write
Mechanism: by reference
The HP DECforms specification for the user in ACMSUDF.DAT.

call_id
Type: ACMSDI_CALL_ID *
Access: write
Mechanism: by reference
A pointer to the call identification returned by the acmsdi_call_task service. To
identify the original task call, compare this pointer with a reference pointer to
the call identifier obtained by acmsdi_return_pointer.

Return Status

The status values returned by the acmsdi_bind_enable_args procedure are
described in Table 4–3.
Table 4–3  acmsdi_bind_enable_args Return Status Values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>Insufficient parameters.</td>
</tr>
<tr>
<td>ACMSDI_INVSUBID</td>
<td>Invalid or obsolete submitter identification.</td>
</tr>
<tr>
<td>ACMSDI_MIXEDMODE</td>
<td>Not a forced nonblocking session.</td>
</tr>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_WRONG_BIND</td>
<td>Trying to bind arguments for an exchange that is not in progress.</td>
</tr>
</tbody>
</table>
acmsdi_bind_msg

4.4 acmsdi_bind_msg

The client application can call this service when `acmsdi_poll` returns `ACMSDI_TDMS_READ_EXCH` (a TDMS Read exchange) or `ACMSDI_TDMS_WRITE_EXCH` (a TDMS Write exchange) from the TP Desktop Connector gateway on the host OpenVMS system. It performs one of the following functions:

- acquires the prompt text, if any, associated with a TDMS Read exchange
- sends the message text associated with a TDMS Read exchange.
- acquires the message text associated with a TDMS Write exchange.

If the prompt or message text is being acquired, the text is truncated when the buffer supplied is not large enough to hold the entire text. If the buffer is larger than the text being acquired, the text is left-justified in the buffer and right-filled with blank characters.

This is an optional call, that is, you are not required to issue this call. However, if you do not issue this call you will not be able to process arguments received from the server or to send arguments back to the server.

Format

```
acmsdi_bind_msg (submitter_id,
                direction,
                length,
                text,
                call_id)
```

Parameters

**submitter_id**
Type: `ACMSDI_SUBMITTER_ID`
Access: `read`
Mechanism: `by reference`
The value returned by the `acmsdi_sign_in` service.

**direction**
Type: `short int`
Access: `read`
Mechanism: `by value`
The value indicates which direction the text is being sent. A value of 1 indicates that the prompt text from a TDMS Read exchange or the message text from a TDMS Write exchange is being copied into the application’s memory.
acmsdi_bind_msg

from ACMS. A value of 0 indicates that the message text for a TDMS Read exchange is being copied to ACMS from the application’s memory.

**length**
Type: **short int**
Access: **read**
Mechanism: **by value**
The length of the text being sent or the length of the buffer to receive the text; specifically one of the following:

- the length of the buffer in the application’s memory which is to receive the prompt text for a TDMS Read exchange.
- the length of the message text being sent to ACMS for a TDMS Read exchange.
- the length of the buffer in the application’s memory which is to receive the message text for a TDMS Write exchange.

**text**
Type: **char**
Access: **read/write**
Mechanism: **by reference**
The text string being sent to ACMS or the buffer which is to hold the text string being acquired from ACMS.

**call_id**
Type: **ACMSDI_CALL_ID **
Access: **write**
Mechanism: **by reference**
Pointer to the call identification returned by the acmsdi_call_task service. To identify the original task call, compare this pointer with a reference pointer to the call identifier obtained by acmsdi_return_pointer.

**Return Status**
This function returns an int value representing a valid TP Desktop Connector status code as defined in ACMSDI.H and described in Table 4–4.
### Table 4-4  acmsdi_bind_msg Return Status Values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>Insufficient parameters.</td>
</tr>
<tr>
<td>ACMSDI_INVSUBID</td>
<td>Invalid or obsolete submitter identification.</td>
</tr>
<tr>
<td>ACMSDI_MIXEDMODE</td>
<td>Not a forced nonblocking session call.</td>
</tr>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_WRONG_BIND</td>
<td>Trying to bind arguments for an exchange that is not in progress.</td>
</tr>
</tbody>
</table>
4.5 acmsdi_bind_receive_args

The client application can call this service whenever an acmsdi_poll returns ACMSDI_RECV_EXCH from the TP Desktop Connector gateway on the OpenVMS system. This service retrieves the write-only arguments passed from the TP Desktop Connector client services. This provides the client application with the receive record identifier and identifies the appropriate set of forms records to send back to ACMS. See also Section 4.6 for information on calls for receive forms records and receive control text. This is an optional call.

**Format**

```c
acmsdi_bind_receive_args (submitter_id,
                          forms_session,
                          receive_record_identifier,
                          receive_record_count,
                          timeout,
                          call_id)
```

**Parameters**

- **submitter_id**
  - Type: ACMSDI_SUBMITTER_ID
  - Access: read
  - Mechanism: by reference
  - The submitter_id returned by the acmsdi_sign_in service.

- **forms_session**
  - Type: ACMSDI_FORMS_SESSION_ID
  - Access: write
  - Mechanism: by reference
  - An identification to associate the session with the form specified in the acmsdi_enable request (see Section 4.11).

- **receive_record_identifier**
  - Type: char *
  - Access: write
  - Mechanism: by reference
  - The form record name or record list name specified in the RECEIVE request in the ACMS task. Refer to *HP TP Desktop Connector for ACMS Client Application Programming Guide* for guidelines on specifying the form name.
acmsdi_bind_receive_args

**receive_record_count**
Type: `long int`
Access: write
Mechanism: by reference
The number of receive record items sent from the ACMS task. The TP Desktop Connector writes the receive_record_count into this location.

**timeout**
Type: `short int`
Access: write
Mechanism: by reference
A timeout value for user input processing sent from the ACMS task. TP Desktop Connector writes the timeout value into this location.

**call_id**
Type: `ACMSDI_CALL_ID *`
Access: write
Mechanism: by reference
A pointer to the call identification returned by the acmsdi_call_task service. To identify the original task call, compare this pointer with a reference pointer to the call identifier obtained by acmsdi_return_pointer.

**Return Status**
The status values returned by the acmsdi_bind_receive_args procedure are described in Table 4–5.

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>Insufficient parameters.</td>
</tr>
<tr>
<td>ACMSDI_INVSUBID</td>
<td>Invalid or obsolete submitter identification.</td>
</tr>
<tr>
<td>ACMSDI_MIXEDMODE</td>
<td>Not a forced nonblocking session call.</td>
</tr>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_WRONG_BIND</td>
<td>Trying to bind arguments for an exchange that is not in progress.</td>
</tr>
</tbody>
</table>
The client application calls this service to send the client application’s receive forms record data to the TP Desktop Connector gateway on the OpenVMS system. Use this service after you have retrieved the exchange step arguments that identify the forms records to be sent to the TP Desktop Connector gateway. This service can also be used to send receive control text to the TP Desktop Connector gateway. This is an optional call.

**Format**

```c
acmsdi_bind_receive_recs (submitter_id,
 receive_control_text_flag,
 receive_record)
```

**Parameters**

- **submitter_id**
  - Type: ACMSDI_SUBMITTER_ID
  - Access: read
  - Mechanism: by reference
  - The value returned by the acmsdi_sign_in service (see Section 2.11).

- **receive_control_text_flag**
  - Type: short integer
  - Access: read
  - Mechanism: by value
  - A value of 1 indicates that receive control text is to be sent to the TP Desktop Connector gateway. A value of 0 indicates that receive control text is not to be sent.

- **receive_record**
  - Type: ACMSDI_FORM_RECORD_BIND array
  - Access: read
  - Mechanism: by reference
  - An array of ACMSDI_FORM_RECORD_BIND structures pointing to buffers that store application data and shadow records to be sent to the ACMS task. (see *HP TP Desktop Connector for ACMS Client Application Programming Guide*). If the send_control_text_flag contains a value of 1, the first ACMSDI_FORM_RECORD_BIND structure in the array must point to the receive control text buffer. No shadow record is associated with receive control text.
acmsdi_bind_receive_recs

Return Status

The status values returned by the acmsdi_bind_receive_recs service are described in Table 4–6.

Table 4–6 acmsdi_bind_receive_recs Return Status Values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>Insufficient parameters.</td>
</tr>
<tr>
<td>ACMSDI_INVSUBID</td>
<td>Invalid or obsolete submitter identification.</td>
</tr>
<tr>
<td>ACMSDI_MIXEDMODE</td>
<td>Not a forced nonblocking session.</td>
</tr>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_WRONG_BIND</td>
<td>Trying to bind arguments for an exchange that is not in progress.</td>
</tr>
</tbody>
</table>
4.7 acmsdi_bind_request_args

The client application can call this service whenever acmsdi_poll returns ACMSDI_REQUEST_EXCH (a TDMS exchange step) from the TP Desktop Connector gateway on the OpenVMS system. This service retrieves the write-only arguments passed from the TP Desktop Connector client services. This provides the client application with the request name and identifies the set of workspaces to be received from and then sent back to ACMS. This is an optional call.

**Format**

```
acmsdi_bind_request_args (submitter_id,
                         request_name,
                         workspace_count,
                         call_id)
```

**Parameters**

- **submitter_id**
  - Type: ACMSDI_SUBMITTER_ID
  - Access: read
  - Mechanism: by reference
  - The value returned by the acmsdi_sign_in service (see Section 2.11).

- **request_name**
  - Type: char *
  - Access: write
  - Mechanism: by reference
  - The name of the TDMS request specified in the ACMS task.

- **workspace_count**
  - Type: long int
  - Access: write
  - Mechanism: by reference
  - The number of workspaces sent from the ACMS task. TP Desktop Connector writes this workspace count into this location.
call_id
Type: ACMSDI_CALL_ID *
Access: write
Mechanism: by reference
The call identification returned by the acmsdi_call_task service. To identify the original task call, compare this pointer with a reference pointer to the call identifier obtained by acmsdi_return_pointer.

Return Status
The status values returned by the acmsdi_bind_request_args service are described in Table 4–7.

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>Insufficient parameters.</td>
</tr>
<tr>
<td>ACMSDI_INVSUBID</td>
<td>Invalid or obsolete submitter identification.</td>
</tr>
<tr>
<td>ACMSDI_MIXEDMODE</td>
<td>Not a forced nonblocking session.</td>
</tr>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_WRONG_BIND</td>
<td>Trying to bind arguments for an exchange that is not in progress.</td>
</tr>
</tbody>
</table>
4.8 acmsdi_bind_request_wksp

The client application calls this service to copy request workspace data between the client application and the ACMS during a TDMS exchange. Use the acmsdi_bind_request_wksp call to copy request workspace data from TP Desktop Connector gateway to the client application memory. After modification, use this call again to copy the modified contents back to the TP Desktop Connector gateway. Use this service after you have retrieved the exchange step arguments that identify the workspaces from the TP Desktop Connector gateway. This is an optional call.

Format

```
acmsdi_enable (submitter_id,
   direction,
   req_wksp_array)
```

Parameters

- **submitter_id**
  - Type: ACMSDI_SUBMITTER_ID
  - Access: read
  - Mechanism: by reference
  - The value returned by the acmsdi_sign_in service (see Section 2.11).

- **direction**
  - Type: short int
  - Access: read
  - Mechanism: by value
  - The value indicates which direction the workspaces are being sent. A value of 1 indicates that the workspaces are being copied into the application’s memory from ACMS. A value of 0 indicates that the workspaces are being copied to ACMS from the application’s memory.

- **workspaces**
  - Type: ACMSDI_WORKSPACE_BIND array
  - Access: read/write
  - Mechanism: by reference
  - The workspace data sent from the ACMS task. One or more workspace arguments can be sent from and returned to the task (see *HP TP Desktop Connector for ACMS Client Application Programming Guide*).
Return Status

The status values returned by the acmsdi_bind_request_wksp service are described in Table 4–8.

Table 4–8 acmsdi_bind_request_wksp Return Status Values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>Insufficient parameters.</td>
</tr>
<tr>
<td>ACMSDI_INVSUBID</td>
<td>Invalid or obsolete submitter identification.</td>
</tr>
<tr>
<td>ACMSDI_MIXEDMODE</td>
<td>Not a forced nonblocking session.</td>
</tr>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_WRONG_BIND</td>
<td>Trying to bind arguments for an exchange that is not in progress.</td>
</tr>
</tbody>
</table>
The client application can call this service whenever an acmsdi_poll returns ACMSDI_SEND_EXCH from the TP Desktop Connector gateway on the OpenVMS system. This service retrieves the write-only arguments passed from the TP Desktop Connector client services. This provides the client application with the send record identifier and identifies the set of forms records it receives from ACMS. See also Section 4.10 for information on calls for send forms records and send control text. This is an optional call.

**Format**

```c
acmsdi_bind_send_args (submitter_id,
                        forms_session,
                        send_record_identifier,
                        send_record_count,
                        timeout,
                        call_id)
```

**Parameters**

- **submitter_id**
  - Type: ACMSDI_SUBMITTER_ID
  - Access: read
  - Mechanism: by reference
  - The submitter_id returned by the acmsdi_sign_in service.

- **forms_session**
  - Type: ACMSDI_FORMS_SESSION_ID
  - Access: write
  - Mechanism: by reference
  - An identification that associates the session with the form specified in the acmsdi_enable request (see Section 4.11).

- **send_record_identifier**
  - Type: char *
  - Access: write
  - Mechanism: by reference
  - The form record name or record list name specified in the SEND request in the ACMS task. Refer to *HP TP Desktop Connector for ACMS Client Application Programming Guide* for guidelines on specifying the form name.
acmsdi_bind_send_args

send_record_count
Type: long int
Access: write
Mechanism: by reference
The number of send record items sent from the ACMS task. TP Desktop Connector writes the send_record_count into this location.

timeout
Type: short int
Access: write
Mechanism: by reference
A timeout value for user input processing, sent from the ACMS task. TP Desktop Connector writes the timeout value into this location.

call_id
Type: ACMSDI_CALL_ID *
Access: write
Mechanism: by reference
A pointer to the call identification returned by the acmsdi_call_task service. To identify the original task call, compare this pointer with a reference pointer to the call identifier obtained by acmsdi_return_pointer.

Return Status

The status values returned by the acmsdi_bind_send_args procedure are described in Table 4–9.

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>Insufficient parameters.</td>
</tr>
<tr>
<td>ACMSDI_INVSUBID</td>
<td>Invalid or obsolete submitter identification.</td>
</tr>
<tr>
<td>ACMSDI_MIXEDMODE</td>
<td>Not a forced nonblocking session.</td>
</tr>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_WRONG_BIND</td>
<td>Trying to bind arguments for an exchange that is not in progress.</td>
</tr>
</tbody>
</table>
4.10 acmsdi_bind_send_recs

The client application calls this service to retrieve send forms record data from the TP Desktop Connector gateway on the OpenVMS system. Use this service after you have retrieved the exchange step arguments that identify the forms records required from the TP Desktop Connector gateway. This service can also be used to retrieve send control text from the TP Desktop Connector gateway. This is an optional call.

Format

```
acmsdi_bind_send_recs (submitter_id,
                       send_control_text_flag,
                       send_record)
```

Parameters

- **submitter_id**
  - Type: ACMSDI_SUBMITTER_ID
  - Access: read
  - Mechanism: by reference
  - The value returned by the acmsdi_sign_in service (see Section 2.11).

- **send_control_text_flag**
  - Type: short integer
  - Access: read
  - Mechanism: by value
  - A value of 1 indicates that send control text is to be copied from the ACMS task. A value of 0 indicates that send control text is not to be copied.

- **send_record**
  - Type: ACMSDI_FORM_RECORD_BIND array
  - Access: write
  - Mechanism: by reference
  - An array of ACMSDI_FORM_RECORD_BIND structures pointing to buffers containing application data and shadow records sent from the ACMS task (see [HP TP Desktop Connector for ACMS Client Application Programming Guide](#)). If the send_control_text_flag contains a value of 1, the first ACMSDI_FORM_RECORD_BIND structure in the array must point to the send control text buffer. There is no shadow record associated with send control text.
acmsdi_bind_send_recs

Return Status

The status values returned by the acmsdi_bind_send_recs service are described in Table 4–10.

Table 4–10  acmsdi_bind_send_recs Return Status Values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>Insufficient parameters.</td>
</tr>
<tr>
<td>ACMSDI_INVSUBID</td>
<td>Invalid or obsolete submitter identification.</td>
</tr>
<tr>
<td>ACMSDI_MIXEDMODE</td>
<td>Not a forced nonblocking session.</td>
</tr>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_WRONG_BIND</td>
<td>Trying to bind arguments for an exchange that is not in progress.</td>
</tr>
</tbody>
</table>
You can issue the acmsdi_bind_session_id service to send the forms session ID argument to ACMS during an enable exchange step. This is an optional call.

**Format**

```
acmsdi_bind_session_id (submitter_id,
            forms_session)
```

**Parameters**

- **submitter_id**
  - Type: `ACMSDI_SUBMITTER_ID`
  - Access: `read`
  - Mechanism: `by reference`
  - The value returned by the acmsdi_sign_in service (see Section 2.11).

- **forms_session**
  - Type: `ACMSDI_FORMS_SESSION_ID`
  - Access: `read`
  - Mechanism: `by reference`
  - An identification that associates the session with the submitter identification.
  - The user-written application can use the `forms_session` parameter to associate the session with the form specified in the enable request. The TP Desktop Connector run-time system passes this parameter to subsequent requests to specify which form to use.
Return Status

The status values returned by the acmsdi_bind_session_id procedure are described in Table 4–11.

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>Insufficient parameters.</td>
</tr>
<tr>
<td>ACMSDI_INVSUBID</td>
<td>Invalid or obsolete submitter identification.</td>
</tr>
<tr>
<td>ACMSDI_MIXEDMODE</td>
<td>Not a forced nonblocking session.</td>
</tr>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_WRONG_BIND</td>
<td>Trying to bind arguments for an exchange that is not in progress.</td>
</tr>
</tbody>
</table>
4.12 acmsdi_bind_transceive_args

The client application can call this service whenever an acmsdi_poll returns ACMSDI_TRCV_EXCH from the TP Desktop Connector gateway on the OpenVMS system. This service retrieves the write-only arguments passed from the TP Desktop Connector client services. This provides the client application with the send and receive record identifiers and identifies the set of forms records to be received from and sent to ACMS. See Section 4.6 for information on calls for receive forms records and receive control text. See Section 4.10 for information on calls for send forms records and send control text. This is an optional call.

**Format**

```c
acmsdi_bind_transceive_args submitter_id,
    forms_session,
    send_record_identifier,
    send_record_count,
    receive_record_identifier,
    receive_record_count,
    timeout,
    call_id )
```

**Parameters**

- **submitter_id**
  - Type: ACMSDI_SUBMITTER_ID
  - Access: read
  - Mechanism: by reference
  - The submitter_id returned by the acmsdi_sign_in service.

- **forms_session**
  - Type: ACMSDI_FORMS_SESSION_ID
  - Access: write
  - Mechanism: by reference
  - An identification to associate the session with the form specified in the acmsdi_enable request (see Section 4.11).
acmsdi_bind_transceive_args

send_record_identifier
Type: char *
Access: write
Mechanism: by reference
The form record name or record list name specified in the SEND request in the ACMS task. Refer to HP TP Desktop Connector for ACMS Client Application Programming Guide for guidelines on specifying the form name.

send_record_count
Type: long int
Access: write
Mechanism: by reference
The number of send record items sent from the ACMS task. TP Desktop Connector writes the send_record_count into this location.

receive_record_identifier
Type: char *
Access: write
Mechanism: by reference
The form record name or record list name specified in the RECEIVE request in the ACMS task. Refer to HP TP Desktop Connector for ACMS Client Application Programming Guide for guidelines on specifying the form name.

receive_record_count
Type: long int
Access: write
Mechanism: by reference
The number of receive record items sent from the ACMS task. TP Desktop Connector writes the receive_record_count into this location.

timeout
Type: short int
Access: write
Mechanism: by reference
A timeout value for user input processing, sent from the ACMS task. TP Desktop Connector writes the timeout value into this location.

call_id
Type: ACMSDI_CALL_ID *
Access: write
Mechanism: by reference
A pointer to the call identification returned by the acmsdi_call_task service. To identify the original task call, compare this pointer with a reference pointer to the call identifier obtained by acmsdi_return_pointer.
Return Status

The status values returned by the acmsdi_bind_transceive_args service are described in Table 4–12.

Table 4–12 acmsdi_bind_transceive_args Return Status Values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>Insufficient parameters.</td>
</tr>
<tr>
<td>ACMSDI_INVSUBID</td>
<td>Invalid or obsolete submitter identification.</td>
</tr>
<tr>
<td>ACMSDI_MIXEDMODE</td>
<td>Not a forced nonblocking session.</td>
</tr>
<tr>
<td>ACMSDI_NORMAL</td>
<td>Normal successful completion.</td>
</tr>
<tr>
<td>ACMSDI_WRONG_BIND</td>
<td>Trying to bind arguments for an exchange that is not in progress.</td>
</tr>
</tbody>
</table>
4.13 **acmsdi_poll**

TP Desktop Connector client programs call this nonblocking service to check for and process messages sent from a TP Desktop Connector gateway to an active submitter in the desktop application. The application must periodically call this service in a forced nonblocking environment to check for completion of outstanding acmsdi_sign_in, acmsdi_call_task, acmsdi_cancel, and acmsdi_sign_out requests. The acmsdi_poll call also checks for the arrival of an exchange step from the back end. (For more information, see *HP TP Desktop Connector for ACMS Client Application Programming Guide*).

The acmsdi_poll service returns a pointer to the call context acquired from the user application when the just-completed call was issued. Storage for this 32-bit integer must exist in the client application's memory. The client application can compare this pointer with that returned by the acmsdi_return_pointer service to identify the completed call. The call context is identified by matching the values.

**Format**

```
acmsdi_poll (submitter_id, call-context)
```

**Parameters**

**submitter_id**
Type: `ACMSDI_SUBMITTER_ID`
Access: **read**
Mechanism: **by reference**
The submitter_id returned by the acmsdi_sign_in service. The submitter identifier is used to identify the submitter for which the call is being issued. The acmsdi_poll service passes the submitter identifier as a read-only argument. TP Desktop responds with either the message type received from the back end for that submitter or a return code indicating that there is no message.

**call_context**
Type: `void *`
Access: **write**
Mechanism: **by reference**
This call writes the pointer to the call context into this optional parameter. This context identifies the call message that has arrived from the back end.
For example, if an `acmsdi_call_task` completion message is sent from the back end, this is a pointer to the context supplied on the `acmsdi_call_task` call.

This pointer to a structure in the client application’s memory is treated as a 32-bit integer. To determine the structure being referenced, the client application compares this value with the values returned by the `acmsdi_return_pointer` service. Because data can be moved by memory management, `acmsdi_return_pointer` calls must be issued within the same procedure as the original call.

### Return Status

The status values returned by the `acmsdi_poll` service are listed in Table 4–13.

**Table 4–13  acmsdi_poll Return Status Values**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ACMSDI_CANCEL_DONE</code></td>
<td>Task cancel call complete.</td>
</tr>
<tr>
<td><code>ACMSDI_DONE</code></td>
<td>Sign-in, sign-out, or task call complete.</td>
</tr>
<tr>
<td><code>ACMSDI_ENABLE_EXCH</code></td>
<td>Enable exchange step has arrived.</td>
</tr>
<tr>
<td><code>ACMSDI_EXCHACTV</code></td>
<td>Request is invalid while exchange step processing is active.</td>
</tr>
<tr>
<td><code>ACMSDI_EXEC</code></td>
<td>No message from the back-end available; call still executing.</td>
</tr>
<tr>
<td><code>ACMSDI_INSUFPRM</code></td>
<td>Insufficient or conflicting parameters.</td>
</tr>
<tr>
<td><code>ACMSDI_INVSUBID</code></td>
<td>Invalid or obsolete submitter identification.</td>
</tr>
<tr>
<td><code>ACMSDI_INTERNAL</code></td>
<td>Internal TP Desktop Connector error.</td>
</tr>
<tr>
<td><code>ACMSDI_MIXEDMODE</code></td>
<td>Not a forced nonblocking session call.</td>
</tr>
<tr>
<td><code>ACMSDI_NOMEMORY</code></td>
<td>Insufficient memory.</td>
</tr>
<tr>
<td><code>ACMSDI_READY</code></td>
<td>No message from the back-end available; no call executing.</td>
</tr>
<tr>
<td><code>ACMSDI_RECV_EXCH</code></td>
<td>Receive exchange step has arrived.</td>
</tr>
<tr>
<td><code>ACMSDI_REQUEST_EXCH</code></td>
<td>TDMS Request exchange step has arrived.</td>
</tr>
<tr>
<td><code>ACMSDI_SEND_EXCH</code></td>
<td>Send exchange step has arrived.</td>
</tr>
</tbody>
</table>
## acmsdi_poll

Table 4–13 (Cont.) acmsdi_poll Return Status Values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_SRVDEAD</td>
<td>TP Desktop Connector gateway is not running on the server node, or the network link has been terminated.</td>
</tr>
<tr>
<td>ACMSDI_TDMS_READ_EXCH</td>
<td>TDMS Read exchange has arrived</td>
</tr>
<tr>
<td>ACMSDI_TDMS_WRITE_EXCH</td>
<td>TDMS Write exchange has arrived</td>
</tr>
<tr>
<td>ACMSDI_TRCV_EXCH</td>
<td>Transceive exchange step has arrived</td>
</tr>
</tbody>
</table>
This chapter describes the system management service available on systems running the TP Desktop Connector gateway. The ACMSDI$GET_SUBMITTER_INFO service returns information about the status of TP Desktop Connector gateway processing on the submitter node.
ACMSDI$GET_SUBMITTER_INFO

5.1 ACMSDI$GET_SUBMITTER_INFO

This service returns information regarding desktop users signed in to the HP ACMS system. The service reports only those users signed in to the TP Desktop Connector gateway running on the same node as the program calling the ACMSDI$GET_SUBMITTER_INFO routine. See HP TP Desktop Connector for ACMS Client Application Programming Guide for coding and building guidelines.

The C-language function prototype and definitions for the item codes are in the file ACMSDI.H in the ACMSDI$COMMON directory.

The ACMSDI$EXAMPLES directory contains a program, SHOW_DESKTOP_USERS.EXE, that uses the ACMSDI$GET_SUBMITTER_INFO service, the source file (.C), and the build command procedure (.COM). See HP TP Desktop Connector for ACMS Client Application Programming Guide for more information about this program.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>A program using the ACMSDI$GET_SUBMITTER_INFO service that was compiled and linked with TP Desktop Connector Version 3.2 works with only TP Desktop Connector Version 3.2.</td>
</tr>
</tbody>
</table>

**Format**

ACMSDI$GET_SUBMITTER_INFO (user_context, itmlst, [target_submitter_ID], [target_desktop_ID], [target_username])

**Parameters**

- **user_context**
  - Type: longword (unsigned)
  - Access: read write
  - Mechanism: by reference
  - A context variable acting as a placeholder while the program reports on multiple submitters. Before calling ACMSDI$GET_SUBMITTER_INFO initially, the program must set the user context variable to zero. On
ACMSDI$GET_SUBMITTER_INFO

ACMSDI$GET_SUBMITTER_INFO calls, if the service returns the status ACMSDI$_NORMAL, the program does not modify the variable value.

Itmlst
Type: longword (unsigned)
Access: read
Mechanism: by reference
Item list describing the information to be reported. Itmlst is the address of a list of item descriptors, each of which specifies or controls an item of information to be returned. The list of item descriptors is terminated by an item code of zero.

Figure 5–1 shows the item descriptor format.

Figure 5–1  Submitter Item Descriptor Format

The valid item codes are described in Table 5–1.

Table 5–1  Submitter Information Item Codes

<table>
<thead>
<tr>
<th>Code Name</th>
<th>Action:</th>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI$K_LATEST_MSG_TIME</td>
<td>Returns the OpenVMS absolute date and time at which the desktop submitter most recently sent a message to the TP Desktop Connector gateway.</td>
<td>The buffer address field of the item descriptor is the address of a quadword in which the ACMSDI$GET_SUBMITTER_INFO service writes this time.</td>
</tr>
</tbody>
</table>

(continued on next page)
Table 5–1 (Cont.) Submitter Information Item Codes

<table>
<thead>
<tr>
<th>Code Name</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI$K_DESKTOP_ID</td>
<td>Returns the desktop gateway submitter identification assigned internally by the TP Desktop Connector gateway.</td>
<td>The buffer address field in the item descriptor is the address of a two-longword structure. ACMSDI$GET_SUBMITTER_INFO writes the desktop gateway submitter number into the first field in the target_desktop_ID array, and the instance into the second field. These two values make up the complete desktop gateway submitter identification.</td>
</tr>
<tr>
<td>ACMSDI$K_NODENAME</td>
<td>Returns the name of the network node from which the desktop submitter is signed in to the ACMS system.</td>
<td>The buffer address field in the item descriptor points to a user-provided buffer into which ACMSDI$GET_SUBMITTER_INFO writes the name. The return length address field of the item descriptor points to a word into which ACMSDI$GET_SUBMITTER_INFO writes the length of the node name in bytes.</td>
</tr>
<tr>
<td>ACMSDI$K_SIGN_IN_TIME</td>
<td>Returns the OpenVMS absolute date and time at which the desktop submitter signed in to the TP Desktop Connector gateway.</td>
<td>The buffer address field of the item descriptor is the address of a quadword in which the ACMSDI$GET_SUBMITTER_INFO service writes this time.</td>
</tr>
</tbody>
</table>

(continued on next page)
ACMSDI$GET_SUBMITTER_INFO

Table 5–1 (Cont.) Submitter Information Item Codes

<table>
<thead>
<tr>
<th>Code Name</th>
<th>Description</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI$K_SUBMITTER_ID</td>
<td></td>
<td>Returns the ACMS submitter_ID of the desktop submitter.</td>
<td>Returns the ACMS submitter_ID of the desktop submitter.</td>
</tr>
<tr>
<td>ACMSDI$K_TRANSPORT</td>
<td></td>
<td>Returns an enumerated longword value corresponding to the transport used for the submitter sign-in.</td>
<td>Returns an enumerated longword value corresponding to the name of the transport used for the submitter sign-in.</td>
</tr>
<tr>
<td>ACMSDI$K_USERNAME</td>
<td></td>
<td>Returns the user name under which the desktop submitter is signed in.</td>
<td>Returns the user name under which the desktop submitter is signed in.</td>
</tr>
</tbody>
</table>

**target_submitter_ID**
Type: unsigned Long Int
Access: read
Mechanism: by reference
The ACMS submitter identification as displayed in the ACMS/SHOW USERS command.

**target_desktop_ID**
Type: unsigned Long Int [2]
Access: read
Mechanism: by reference
The desktop gateway submitter identification, ACMS$DESKTOP_ID, on which to report. The first long int contains the desktop gateway submitter number and the second long int contains the instance.
ACMSDI$GET_SUBMITTER_INFO

target_username
Type: character string descriptor
Access: read
Mechanism: by descriptor
The user name on which to report.

Return Status

The status values returned by the ACMSDI$GET_SUBMITTER_INFO service are listed in Table 5–2.

Table 5–2   ACMSDI$GET_SUBMITTER_INFO Return Status Values

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI$_ILLITEMCODE</td>
<td>An illegal item code appears in the item list. No information is reported.</td>
</tr>
<tr>
<td>ACMSDI$_NOMATCHSUBS</td>
<td>No matching submitter. No submitter matches the selection criteria.</td>
</tr>
<tr>
<td>ACMSDI$_NOMORESUBS</td>
<td>No more submitters to report. No information was reported by this call, because no more submitters match the selection criteria. The snapshot is consistent with the current set of submitters.</td>
</tr>
<tr>
<td>ACMSDI$_NORMAL</td>
<td>Normal successful completion. Information specified by the item list has been reported about a matching submitter. Additional matching submitters can remain to be reported.</td>
</tr>
<tr>
<td>ACMSDI$_OBSINFOREP</td>
<td>Obsolete information reported. No information was returned, because no more submitters match the selection criteria. The snapshot is not consistent with the current set of submitters.</td>
</tr>
</tbody>
</table>
Data Compression Monitor Commands

This chapter provides a description of the Data Compression Monitor commands that you can use to monitor compression activity. You can shorten all commands and keywords to the smallest unambiguous abbreviation, which is at most three characters.

See *HP TP Desktop Connector for ACMS Client Application Programming Guide* for more information on using the Data Compression Monitor.
6.1 EXIT

This command exits the Compression Monitor Activity reporting program.

Format

`EXIT`
6.2 HELP

Displays the help file, SYS$HLP:ACMSDI$DCM_REPORTER_HLP.TXT.

Format

HELP
6.3 LIST

This command generates a report, which you can display on the screen or write to a file.

Format

LIST [/[qualifier]]

Qualifiers

/APPLICATION=application
Allows you to select detailed records associated with the application(s) specified. The application specification can be an ACMS application name or a list of application names. If you specify a list, separate the names with a comma and enclose the list within parentheses. The asterisk (*) wild card character is permitted in application name specifications. If you do not specify /APPLICATION, records for all applications are selected unless you set default application(s) with the SET command.

Specifying /APPLICATION with the LIST command overrides any default applications that you previously set. Specifying /APPLICATION=* on the LIST command selects records for all applications, overriding any defaults that are set.

/BEFORE[=date-time]
Selects detailed records that were written before the date and time specified. The date-time specification is optional. If you omit it, records written earlier than the current date are selected. Specify date and time in the OpenVMS standard format dd-mmm-yyyy:hh-mm-ss.

/DETAILS
Specifies that the report should contain details of all calls. This is the default, unless you specify /SUMMARY in the SET command. Specifying /DETAILS with the LIST command overrides SET/SUMMARY.

/INPUT=file
Specifies a source file from which records for the report are to be read. If you do not specify a source file, the latest version of SYS$ERRORLOG:ACMSDI$COMPRESSION.LOG is used, unless you have previously specified a default input file using the SET command. To override a default input file setting, use /INPUT=* on the LIST command.
LIST

/NODE=(node-identifier[,...])
Selects detailed records associated with task calls originating from the client node(s) specified. The node-identifier can be a DECnet node name, TCP/IP address, or a list of same. If you specify a list, separate the identifiers with commas and enclose the list within parentheses. If you specify a single identifier, you do not need the comma and parentheses. The asterisk (*) wild card character is permitted in node identifiers. If you do not specify /NODE, records for all nodes are selected unless you set default node(s) with the SET command.

Specifying /NODE with the LIST command overrides any default nodes that you may have set. Specifying /NODE= with the LIST command specifies that records for all nodes are to be selected, overriding any defaults that you may have set.

/OUTPUT=file
Directs the report to a file. The file specification must be a valid OpenVMS file specification. Displaying the report on the screen is the default, unless you specify the /OUTPUT qualifier with the SET command. Specifying /OUTPUT=* with the LIST command, displays any reports on the screen, overriding any defaults that may be set.

/SINCE=[date-time]
Selects detailed records that were written on or after the date and time specified. The date-time specification is optional. If you omit it, the records written on the current date are selected. Specify the date and time in the OpenVMS format, dd-mmm-yyyy:hh:mm:ss.

/SUMMARY
Specifies that a summary report, omitting details, be written. The default is a detailed report containing all calls be written. You can set your own default with the SET command. Summary reports show totals of the uncompressed workspace sizes, the compressed workspace sizes, and the number of bytes saved by data compression.

/TASK=task-name
Selects detailed records associated with ACMS task calls for the task name(s) specified, including any exchange steps associated with the task calls. The task-name specification can be a valid ACMS task name or a list of task names. If you specify a list, separate the names with a comma and enclose the list within parentheses. You can use the asterisk (*) wild-card character in the task-name specification. If you do not specify the /TASK qualifier, records for all tasks are selected, unless you have specified another default with the SET command. If you specify /TASK=* with the LIST command, records for
LIST

all tasks are selected. If tasks specified are part of more than one ACMS application, matching task details for all applications are selected, unless you narrow the selection further with the /APPLICATION qualifier.

/USER=user-identifier
Selects detailed records associated with ACMS task calls executed for the signed-in user session(s) specified. The user-identifier specification can be a user identifier or a list of user identifiers. If you specify a list, separate the identifiers with a comma and enclose the list within parentheses. The asterisk (*) wild card character is permitted in user identifier specifications. If /USER is not specified, records for all users are selected, unless default user identifier(s) have been selected with the SET command. Specifying /USER on the LIST command overrides any default user identifiers that may have been set. Specifying /USER=* on the LIST command specifies that records for all user sessions are selected, overriding any defaults that may have been set.

Examples

1. /APPLICATION=LARRY
   Selects records for application LARRY.

2. /APPLICATION=(KURT, SARAH)
   Selects records for applications KURT and SARAH.

3. /APPLICATION=*DEC*
   Selects records for applications that contain DEC in their names.

4. /BEFORE
   Selects records written yesterday and earlier.

5. /BEFORE=12-JUN
   Selects records written prior to June 12th of this year.

6. /BEFORE=16:30
   Selects records written prior to today at 4:30 p.m.

7. /BEFORE=12-JUN-2002:9:15:30
   Selects records written prior to 30 seconds after 9:15 a.m. on June 12, 2002.
8. /NODE=ALPHA1
   Selects all records for task calls from node ALPHA1.

9. /NODE=(LION,TIGER,PANTHR)
   Selects all records for task calls from nodes LION, TIGER, and PANTHR.

10. /NODE=*CPQ*
    Selects all records for task calls from nodes that contain CPQ in their names.

11. /OUTPUT=DAILY_COMPRESSION.REP
    Directs the report to a file named DAILY_COMPRESSION.REP in the current directory.

12. /SINCE
    Selects records written today (after midnight yesterday, or 0:00 today).

13. /SINCE=13-JUN
    Selects records written after midnight, June 12th of the current year.

14. /SINCE=11:25
    Selects records written at or after 11:25 this morning.

15. /SINCE=12-OCT-2001:8:0:45
    Selects records written at or after 45 seconds after 8 a.m. on October 12, 2001.

16. /TASK=HYACINTH
    Selects records for task calls for ACMS task HYACINTH.

17. /TASK=(APRIL,MAY,JUNE)
    Selects records for task calls for ACMS tasks APRIL, MAY, and JUNE.

18. /TASK=*DEF
    Selects all records for task calls associated with tasks with names ending in DEF.
LIST

19. /USER=MAIN-PLANT
   Selects all records for sessions established for user identifier MAIN-PLANT.

20. /USER=(SCHMIDT, MASELLA, RAJIV, SWEENEY)
   Selects all records for sessions established for user identifiers SCHMIDT, MASELLA, RAJIV, and SWEENEY.

21. /USER=SITE3*
   Selects all records for sessions established for user identifiers that begin with SITE3.
6.4 RENEW

This command closes the current log file and opens a new one. The new log file is a new version of ACMSDI$COMPRESSION.LOG.

Format

RENEW
SELECT

6.5 SELECT

This command selects records from the log file and writes them to a file from which you can generate customized reports. If you do not specify a qualifier, all records are selected.

Format

SELECT file [/qualifier]

Parameters

file
Is a required parameter that specifies the name of the file to which the selected records are written.

Qualifiers

/APPLICATION=application
Allows you to select detailed records associated with the application(s) specified. The application specification can be an ACMS application name or a list of application names. If you specify a list, separate the names with a comma and enclose the list within parentheses. The asterisk (*) wild card character is permitted in application name specifications. If you do not specify /APPLICATION, records for all applications are selected unless you set default application(s) with the SET command.

Specifying /APPLICATION with the SELECT command overrides any default applications that you previously set. Specifying /APPLICATION=* on the SELECT command selects records for all applications, overriding any defaults that are set.

/BEFORE=[date-time]
Selects detailed records that were written before the date and time specified. The date-time specification is optional. If you omit it, records written earlier than the current date are selected. Specify date and time in the OpenVMS standard format dd-mmm-yyyy:hh-mm-ss.

/INPUT=file
Specifies a source file from which records for the report are to be read. If you do not specify a source file, the latest version of SYS$ERRORLOG: ACMSDI$COMPRESSION.LOG is used, unless you have previously specified
a default input file using the SET command. To override a default input file setting, use /INPUT=* on the LIST command.

/NODE=(node-identifier[,...])
Selects detailed records associated with task calls originating from the client node(s) specified. The node-identifier can be a DECnet node name, TCP/IP address, or a list of same. If you specify a list, separate the identifiers with commas and enclose the list within parentheses. If you specify a single identifier, you do not need the comma and parentheses. The asterisk (*) wild card character is permitted in node identifiers. If you do not specify /NODE, records for all nodes are selected unless you set default node(s) with the SET command.

Specifying /NODE with the SELECT commands overrides any default nodes that you may have set. Specifying /NODE=* with the SELECT commands specifies that records for all nodes are to be selected, overriding any defaults that you may have set.

/SINCE[=date-time]
Selects detailed records that were written on or after the date and time specified. The date-time specification is optional. If you omit it, the records written on the current date are selected. Specify the date and time in the OpenVMS format, dd-mmm-yyyy:hh:mm:ss.

/TASK=task-name
Selects detailed records associated with ACMS task calls for the task name(s) specified, including any exchange steps associated with the task calls. The task-name specification can be a valid ACMS task name or a list of task names. If you specify a list, separate the names with a comma and enclose the list within parentheses. You can use the asterisk (*) wild-card character in the task-name specification. If you do not specify the /TASK qualifier, records for all tasks are selected, unless you have specified another default with the SET command. If you specify /TASK=* with the SELECT command, records for all tasks are selected. If tasks specified are part of more than one ACMS application, matching task details for all applications are selected, unless you narrow the selection further with the /APPLICATION qualifier.

/USER=user-identifier
Selects detailed records associated with ACMS task calls executed for the signed-in user session(s) specified. The user-identifier specification can be a user identifier or a list of user identifiers. If you specify a list, separate the identifiers with a comma and enclose the list within parentheses. The asterisk (*) wild card character is permitted in user identifier specifications. If /USER is not specified, records for all users are selected, unless default user
SELECT

identifier(s) have been selected with the SET command. Specifying /USER with the SELECT command overrides any default user identifiers that may have been set. Specifying /USER=* with the SELECT command specifies that records for all user sessions are selected, overriding any defaults that may have been set.
6.6 SET

This command sets default values for the LIST and SELECT commands. All qualifiers, except /BEFORE and /SINCE, can have a default value. You can use the SHOW command to display the current default settings. When you set a default, it applies to all reports you request with the LIST command and to all records you select with the SELECT command. However, you can override these default settings on the command line by specifying certain qualifiers.

Format

SET /qualifier

Qualifiers

/APPLICATION=application
Allows you to select detailed records associated with the application(s) specified. The application specification can be an ACMS application name or a list of application names. If you specify a list, separate the names with a comma and enclose the list within parentheses. The asterisk (*) wild card character is permitted in application name specifications. If you do not specify /APPLICATION, records for all applications are selected.

/DETAILS
This default is applicable to the LIST command only. It specifies that the report is to contain detailed information for all calls. This is the standard default.

/INPUT=file
Specifies a source file from which records for the report are to be read. If you do not specify a source file, the latest version of SYS$ERRORLOG: ACMSDI$COMPRESSION.LOG is used, unless you have previously specified a default input file using the SET command. To override a default input file setting, use /INPUT=* on the LIST command.

/NODE=(node-identifier[,...])
Selects detailed records associated with task calls originating from the client node(s) specified. The node-identifier can be a DECnet node name, TCP/IP address, or a list of same. If you specify a list, separate the identifiers with commas and enclose the list within parentheses. If you specify a single identifier, you do not need the comma and parentheses. The asterisk (*) wild card character is permitted in node identifiers.
SET

/OUTPUT=file
Directs the report to a file. The file specification must be a valid OpenVMS file specification. Displaying the report on the screen is the default, unless you specify the /OUTPUT qualifier with the SET command. Specifying /OUTPUT=* with the LIST command, displays any reports on the screen, overriding any defaults that may be set.

/SUMMARY
Specifies that a summary report, omitting details, be written. The default is a detailed report containing all calls be written. You can set your own default with the SET command. Summary reports show totals of the uncompressed workspace sizes, the compressed workspace sizes, and the number of bytes saved by data compression.

/TASK=task-name
Selects detailed records associated with ACMS task calls for the task name(s) specified, including any exchange steps associated with the task calls. The task-name specification can be a valid ACMS task name or a list of task names. If you specify a list, separate the names with a comma and enclose the list within parentheses. You can use the asterisk (*) wild-card character in the task-name specification. If you specify /TASK=*, records for all tasks are selected. If tasks specified are part of more than one ACMS application, matching task details for all applications are selected, unless you narrow the selection further with the /APPLICATION qualifier.

/USER=user-identifier
Selects detailed records associated with ACMS task calls executed for the signed-in user session(s) specified. The user-identifier specification can be a user identifier or a list of user identifiers. If you specify a list, separate the identifiers with a comma and enclose the list within parentheses. The asterisk (*) wild card character is permitted in user identifier specifications. Specifying /USER=*, selects records for all users.
SHOW

6.7 SHOW

This command displays the default values for qualifiers of the LIST and SELECT commands, which you have set with the SET command. All command qualifiers, except /BEFORE and /SINCE, can have a default value. Specifying SHOW without any qualifiers displays all defaults.

Format

SHOW [/qualifier]

Qualifiers

/APPLICATION
Displays the default for /APPLICATION qualifier.

/DETAILS
Displays /DETAILS if /DETAILS is the default; displays /SUMMARY if /SUMMARY is the default. /DETAILS and /SUMMARY are mutually exclusive.

/INPUT
Displays the default for the /INPUT qualifier.

/NODE
Displays the default for the /NODE qualifier.

/OUTPUT
Displays the default for the /OUTPUT qualifier.

/SUMMARY
Displays /SUMMARY if /SUMMARY is the default; displays /DETAILS if /DETAILS is the default. /SUMMARY and /DETAILS are mutually exclusive.

/TASK
Displays the default for the /TASK qualifier.

/USER
Displays the default for the /USER qualifier.
Table A–1 lists the ACMS system status values and their corresponding numeric values as defined in ACMSDI.H and returned in the *err2* parameter, with corresponding symbols.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_NORMAL</td>
<td>0</td>
<td>Normal completion</td>
</tr>
<tr>
<td>ACMSDI_APPLDEAD</td>
<td>–3001</td>
<td>ACMS application not started</td>
</tr>
<tr>
<td>ACMSDI_CALLACTV</td>
<td>–3002</td>
<td>Call active — cannot start new operation</td>
</tr>
<tr>
<td>ACMSDI_INSUFPRM</td>
<td>–3003</td>
<td>Insufficient parameters</td>
</tr>
<tr>
<td>ACMSDI_INTERNAL</td>
<td>–3004</td>
<td>Internal error</td>
</tr>
<tr>
<td>ACMSDI_INVCALLID</td>
<td>–3005</td>
<td>Invalid call identification</td>
</tr>
<tr>
<td>ACMSDI_INVLOGIN</td>
<td>–3006</td>
<td>Invalid login attempt</td>
</tr>
<tr>
<td>ACMSDI_INVOPTION</td>
<td>–3007</td>
<td>Invalid submitter option</td>
</tr>
<tr>
<td>ACMSDI_INVSUBID</td>
<td>–3008</td>
<td>Invalid submitter identification</td>
</tr>
<tr>
<td>ACMSDI_MIXEDMODE</td>
<td>–3009</td>
<td>Using both blocking &amp; non-blocking modes</td>
</tr>
<tr>
<td>ACMSDI_NOACMS</td>
<td>–3010</td>
<td>ACMS not active</td>
</tr>
<tr>
<td>ACMSDI_NOMEMORY</td>
<td>–3011</td>
<td>Low memory resource</td>
</tr>
<tr>
<td>ACMSDI_NOPPACTV</td>
<td>–3012</td>
<td>No active presentation procedure</td>
</tr>
<tr>
<td>ACMSDI_NOSUCH_APPL</td>
<td>–3013</td>
<td>ACMS application not found</td>
</tr>
<tr>
<td>ACMSDI_NOSUCH_TASK</td>
<td>–3014</td>
<td>Invalid task code</td>
</tr>
<tr>
<td>ACMSDI_OPR_CANCE</td>
<td>–3015</td>
<td>Operator canceled ACMS user</td>
</tr>
</tbody>
</table>

(continued on next page)
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACMSDI_PENDING</td>
<td>-3016</td>
<td>Operation started</td>
</tr>
<tr>
<td>ACMSDI_SECCHK</td>
<td>-3017</td>
<td>ACMS task ACL failure</td>
</tr>
<tr>
<td>ACMSDI_SIGNINACTV</td>
<td>-3018</td>
<td>Sign-in in process</td>
</tr>
<tr>
<td>ACMSDI_SIGNOUTACTV</td>
<td>-3019</td>
<td>Sign-out in process</td>
</tr>
<tr>
<td>ACMSDI_SRVDEAD</td>
<td>-3020</td>
<td>TP Desktop Connector server has died</td>
</tr>
<tr>
<td>ACMSDI_TASK_ABORT</td>
<td>-3021</td>
<td>Task has aborted</td>
</tr>
<tr>
<td>ACMSDI_TASK_CANCELLED</td>
<td>-3022</td>
<td>Task canceled by operator</td>
</tr>
<tr>
<td>ACMSDI_TASK_SP_DIED</td>
<td>-3023</td>
<td>Task procedure server process has died</td>
</tr>
<tr>
<td>ACMSDI_TASK_FAILED</td>
<td>-3024</td>
<td>Task failed to complete normally</td>
</tr>
<tr>
<td>ACMSDI_INVPROTOCOL</td>
<td>-3025</td>
<td>Protocol versions of the DDEV and the TP Desktop Connector server do not match</td>
</tr>
<tr>
<td>ACMSDI_BADNODENAME</td>
<td>-3026</td>
<td>Invalid node name</td>
</tr>
<tr>
<td>ACMSDI_PWDexpired</td>
<td>-3027</td>
<td>Password has expired</td>
</tr>
<tr>
<td>ACMSDI_CANCELCANCELACTV</td>
<td>-3028</td>
<td>Client-initiated cancel in progress</td>
</tr>
<tr>
<td>ACMSDI_EXCHACTV</td>
<td>-3029</td>
<td>User-written presentation procedure not completed</td>
</tr>
<tr>
<td>ACMSDI_DISPATCHACTV</td>
<td>-3030</td>
<td>ACMSDI_DISPATCH_MESSAGE call in process</td>
</tr>
<tr>
<td>ACMSDI_UNSUPPORTED</td>
<td>-3031</td>
<td>Unsupported option requested on acmsdi_sign_in or acmsdi_call_task</td>
</tr>
<tr>
<td>ACMSDI_PWD_EXPIRING</td>
<td>-3100</td>
<td>Number of hours returned until password expires</td>
</tr>
<tr>
<td>ACMSDI_CALL_CANCELED</td>
<td>-3101</td>
<td>The task was canceled by the task submitter</td>
</tr>
</tbody>
</table>
Descriptions of client messages and server messages are provided in the following files:

- SYS$HELP:ACMSDI$CLIENT_MESSAGES.TXT
- SYS$HELP:ACMSDI$SERVER_MESSAGES.TXT
Access
  in documentation format, 1–3
  parameter, 1–1
ACMS$DESKTOP_ID submitter
description, 5–5
ACMSDI$GET_SUBMITTER_INFO service
description, 5–2
  sample program using, 5–2
acmsdi_bind_enable_args routine
description of interface, 4–7
acmsdi_bind_msg routine, 4–10
acmsdi_bind_receive_args routine
description of interface, 4–13
acmsdi_bind_receive_recs routine
description of interface, 4–15
acmsdi_bind_request_args routine
description of interface, 4–17
acmsdi_bind_request_wksp
  description of interface, 4–19
acmsdi_bind_send_args routine
description of interface, 4–21
acmsdi_bind_send_recs routine
description of interface, 4–23
acmsdi_bind_session_id routine
description of interface, 4–25
acmsdi_bind_transceive_args routine
description of interface, 4–27
ACMSDI_CALL_ID data type, 1–2
ACMSDI_CALL_OPTIONS data type
  structure, 1–2
acmsdi_call_task service
description, 2–13
acmsdi_cancel service
description, 2–18
acmsdi_check_version routine
  interface description, 3–25
acmsdi_complete_call routine
description of interface, 4–4
acmsdi_complete_pp service
description, 2–21
acmsdi_disable routine
description of interface, 3–5
acmsdi_dispatch_message service
description, 2–23
acmsdi_enable routine
description of interface, 3–6
ACMSDI_FORMS_SESSION_ID data type
  structure, 1–2
ACMSDI_FORM_RECORD data type
  definition, 3–3
  structure, 1–2
ACMSDI_FORM_RECORD_BIND data type
  definition, 4–2
  structure, 1–2
ACMSDI_FORM_RECORD_BIND structure, 2–7
acmsdi_get_version routine
  interface description, 3–27
ACMSDI_INIT_FORM_RECORD data type
  definition, 3–3
ACMSDI_INIT_FORM_RECORD_BIND data
type
  definition, 4–2
ACMSDI_INIT_WORKSPACE data type
  definition, 2–4
ACMSDI_OPTION array
  using, 2–11
ACMSDI_OPTION data type
  structure, 1–2
ACMSDI_OPT_CHECK_VERSION option
  defined, 2–10
  example, 2–27
ACMSDI_OPT_COMMID option
  defined, 2–10
  example, 2–11
ACMSDI_OPT_END_LIST option
  defined, 2–10
ACMSDI_OPT_FREE_ROUTINE option
  defined, 2–10
ACMSDI_OPT_MALLOC_ROUTINE option
  defined, 2–10
ACMSDI_OPT_NONBLK option
  defined, 2–10
ACMSDI_OPT_PWD_EXPIRING option
  defined, 2–10
acmsdi_poll service
  description, 4–30
acmsdi_read_msg function, 3–9
acmsdi_receive routine
  description of interface, 3–11
acmsdi_request routine
  description of interface, 3–14
acmsdi_return_pointer service
  description, 2–25
acmsdi_send routine
  description of interface, 3–16
acmsdi_sign_in service
  description, 2–26
acmsdi_sign_out service
  description, 2–29
ACMSDI_SUBMITTER_ID data type
  structure, 1–2
ACMSDI_SUBMITTER_ID option
  description, 2–27
ACMSDI_TCPIP_PORT_host_node variable, 2–11
acmsdi_transceive routine
  description of interface, 3–19
ACMSDI_WORKSPACE data type
  definition, 2–4
  structure array, 1–2
ACMSDI_WORKSPACE_BIND data type
  definition, 4–3
  structure, 1–2
acmsdi_write_msg routine, 3–23

B
Blocking service
  presentation procedure usage, 3–4
  restriction, 2–3, 3–4
  specifying, 2–2
Brackets
  square, in format, 1–1

C
Call_id parameter
  specification, 2–21
Client service
  summary, 2–1
Completion routine
  format, 2–3
  specifying, 2–2

D
Data compression monitor commands, 6–1
Data type
  parameter, 1–2

E
EXIT command
  description, 6–2
Forced nonblocking services
   described, 4–1
   specifying, 2–3
   summary, 4–1

Form record
   initialization macro, 3–3
   type definition, 3–3, 4–2

Gateway
   task cancellation status, 2–15, 4–4

HELP command
   description, 6–3

LIST command
   description, 6–4

Management
   service, 5–1
   Mechanism
      parameter, 1–1
      parameter passing, 1–3
   Memory
      allocating parameters, 3–3
   Memory allocation, 2–2
   Modify
      access method, 1–3

Nonblocking service
   See also Forced nonblocking presentation procedure usage, 3–4
   restriction, 2–3, 3–4
   specifying, 2–2

OpenVMS system
   management service, 5–1
   Options
      specifying sign-in, 2–27

Parameter
   allocating memory, 3–3
   data type, 1–2
   passing mechanism, 1–3
   Portable API client services, 2–1
   Portable API presentation procedure
      summary, 3–1
   Presentation procedure
      status expected, 3–2

Read
   access method, 1–3
   RENEW command
      description, 6–9
   Return status
      description, 1–4

SELECT command
   description, 6–10
   Server
      See Gateway
   Service
      client, 2–1
      OpenVMS
         management, 5–1
         summary, 2–1
   Service description
      documentation format, 1–1
   Session
      forms identification, 3–6, 4–25
Session Environments
  description, 1–4
SET command
  description, 6–13
SHOW command
  description, 6–15
SHOW_DESKTOP_USERS program
  location, 5–2
Sign-in
  service description, 2–26
Square brackets
  use in format, 1–1
Status
  return, 1–4
Submitter
  identification
    ACM, 5–5
    desktop gateway, 5–4
    program for information, 5–2
Submitter_id
  description, 2–27
System status values, A–1

T
Target desktop

V
Version checking
  example, 2–27
  routine
    description, 3–25

W
Workspace
  defining multiple, 2–5
  initialization macro, 2–4
  relocation by memory management, 4–31
  structure definition, 2–4, 4–3
Write
  access method, 1–3