HP TP Desktop Connector

Getting Started

January 2006

This document introduces you to the Adapter Interface of the *HP TP Desktop Connector* product and provides information for building client applications that call ACMS tasks from various Automation, Java, and C-language interfaces. This guide also describes managing the environment in which the client applications run.

Contents

7 Writing Automation Servers and Clients

8 Writing Java Clients

9 Managing the Client Interface

A TPware Error Logging

Index

Examples

Figures

Tables

Preface

This document provides reference and usage information that enables you to use the *HP TP Desktop Connector* product.

Intended Audience

This document is intended for the following audiences:

- Application programmers who need reference information about developing client programs with the HP TP Desktop Connector product to call ACMS tasks from various Java, Automation, and C-language interfaces.

- Anyone who is responsible for configuring and managing a client interface developed with the HP TP Desktop Connector product.

Operating System Information

For information about the versions of the operating system and other software that are compatible with this version of the HP TP Desktop Connector product, refer to the product's *Software Product Description* (SPD). Use the SPD to verify which versions of your operating system are compatible with this version of the HP TP Desktop Connector product.

Structure

This document has the components shown in the following table:

Component	Description
Chapter 1	Provides an overview of how the HP TP Desktop Connector product allows client systems to call HP transaction processing (TP) applications.
Chapter 2	Introduces the general procedures for building TPware interfaces for client systems to access ACMS tasks.

Component	Description
Chapter 3	Describes HP TP Desktop Connector components used for development on the ACMS system and for runtime support and management on a Windows system. Explains the use of the development components to translate ACMS applications so that you can build a client interface to access the ACMS applications.
Chapter 4	Describes the STDL compiler option flags used in generating HP TP Desktop Connector client programs.
Chapter 5	Explains how to use the TPware client build utility to create a client interface.
Chapter 6	Describes writing and building C and asynchronous clients.
Chapter 7	Discusses writing Automation servers and clients.
Chapter 8	Describes steps for writing and building Java clients.
Chapter 9	Explains how to use the TPware management utility on a Windows system to customize runtime settings for connecting HP TP Desktop Connector client programs that run on a Windows system and ACMS applications.
Appendix A	Describes TPware error logging support.

Associated Documents

Documents associated with the Desktop Client Services Interface are:

- *HP TP Desktop Connector for ACMS* *Client Application Programming Guide*

- *HP TP Desktop Connector for ACMS* *Client Services Reference Manual*

- *HP TP Desktop Connector for ACMS* *Gateway Management Guide*

Documents associated with the installation of this product and related options are:

- *HP TP Desktop Connector for ACMS* *Installation Guide*

Conventions

This document uses the following conventions:

Convention	Description
italic type	Italic type emphasizes important information and indicates complete titles of documents. In format descriptions, it indicates variable parts of input that you must supply.
bold type	New terms are highlighted in boldface type where they are defined in the text.
`user input`	In examples, user input is differentiated in boldface font from system output.
`monospace text`	Words in monospaced font in text indicate names of files, commands and options, user interface elements such as menu names, objects stored on the system, or system output used in examples.
lowercase	Commands that you enter (except where the command line includes an environment variable) are in lowercase.
UPPERCASE	Uppercase text indicates STDL syntax. Environment variables are in uppercase. Key words on the OpenVMS system are also indicated by uppercase text.
[]	In command syntax, square brackets enclose an optional choice or choices.
. . .	In examples, a vertical ellipsis indicates 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 HP TP Desktop Connector product, see the product's *Software Product Description* (SPD).

1

Introduction

This chapter provides an overview of the adapter technology used by TPware software and, in particular, the capabilities of the *HP TP Desktop Connector* product's Adapter Interface.

1.1 TPware Products

TPware is a set of components from which several TP-related products are created. Different components are used to create different products. These products are as follows:

- *HP TP Desktop Connector*
- *HP TP Web Connector*

This manual describes the HP TP Desktop Connector product, which allows single-user client systems to call TP applications.

The HP TP Desktop Connector product provides the following interfaces (see Section 1.6 for an overview):

- TPware Adapter Interface (described in this manual)
- Client Services Interface for ACMS applications (described in other manuals; see "Associated Documents" in the Preface)

1.2 Transaction Processing Through TPware

Typically, transaction processing (TP) applications connect different systems that handle the user interface, the business logic, and the database access functions of a business solution. For applications that need to be connected to clients over a network, the interface is usually a PC client. Through its adapter technology, TPware provides an interface between clients running on a PC and the business application.

TPware can connect dissimilar clients and servers using a variety of communications mechanisms. You can use TPware to write client applications that call ACMS tasks. You can write these client applications either in C or Java, or by using Microsoft Automation.

1.3 Connecting a Client Application to TP Applications

The HP TP Desktop Connector product allows you to connect a client application to a TP server application. The basic model that the HP TP Desktop Connector product uses to connect single-user client systems to TP applications is illustrated in Figure 1–1.

Figure 1–1   Connecting a Client Application to a TP Application
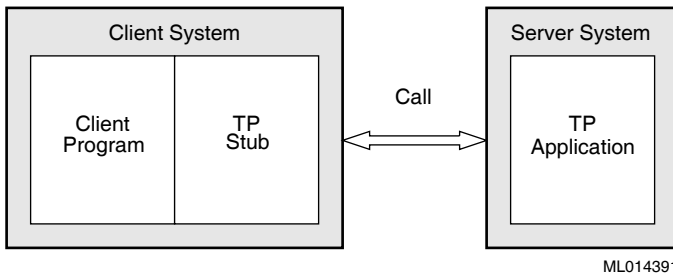
ML014391

TPware accomplishes client access to TP applications by enabling the creation of code called a **TP stub** that translates calls between client programs and TP applications.

Using a client program, the user enters a request by entering information in a programmer-designed window. The client program processes the incoming request and extracts information needed to call the TP application.

The client program then performs the following operations:

- Calls a TP application through the TP stub.

- Interprets the results from the TP application.

- Displays those results to the end user.

The TP application is a TDL task executing under ACMS on an OpenVMS platform system.

1.4 Stub Generation and Adapters

TPware uses TP stub technology to support the generation of interfaces that allow PC-based clients to call tasks using a variety of language interfaces and communications mechanisms. For example, Figure 1–2 shows how a TP stub allows two types of interfaces to call each other.

Figure 1–2   TP Stub and Adapter Technology
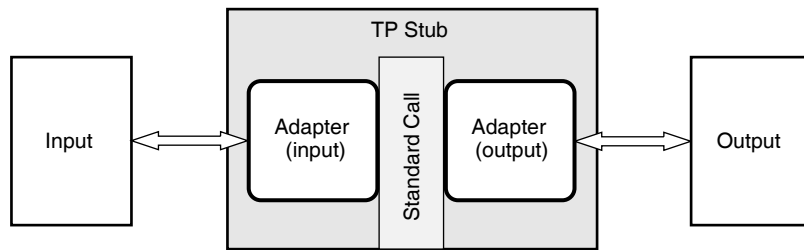
ML014583

To accomplish this stub-based calling mechanism, TPware divides the functionality of the TP stub into halves called **adapters**. One half of the TP stub, the **input adapter**, takes an incoming call and transforms it into a standard form called the **TPware standard call**. The other half, the **output adapter**, takes the TPware standard call and transforms it into the appropriate form for the TP application.

In this way, TPware software extends the capabilities of stubs. By supporting stubs with various types of adapters, TPware software can support different language interfaces and communications mechanisms. Figure 1–3 illustrates the types of adapters that a HP TP Desktop Connector system supports.

The HP TP Desktop Connector software generates and uses the following types of input adapters:

* Automation input
* Java input (Java/JavaBeans)
* C input
* Asynchronous C input

Figure 1–3 Adapters Supported by HP TP Desktop Connector
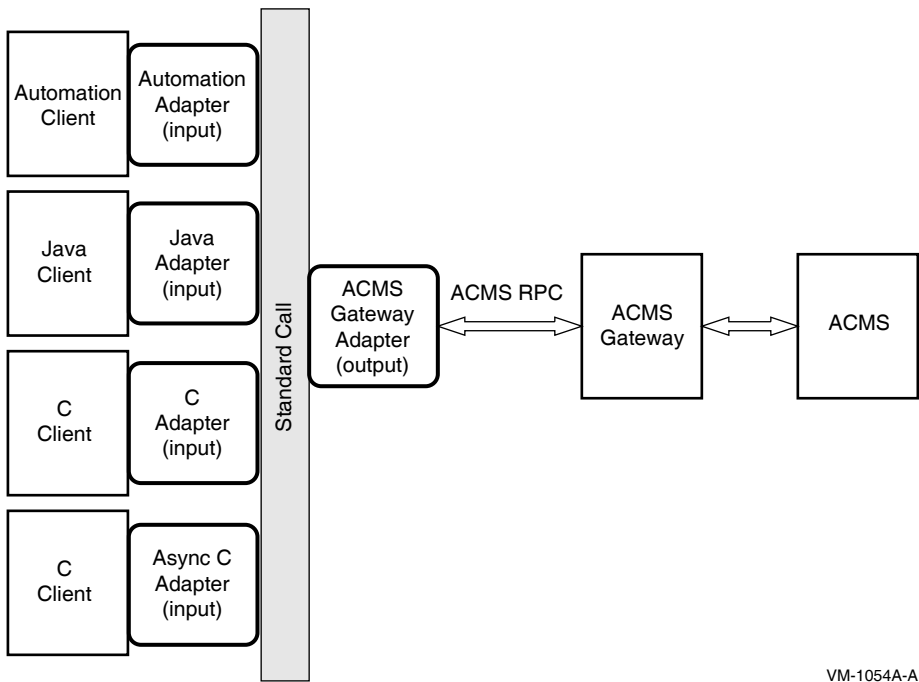
VM-1054A-AI

The HP TP Desktop Connector software generates and uses the following types of output adapters:

- ACMS Gateway output (for calls to ACMS tasks)

1.5 Adapter Behavior

For the HP TP Desktop Connector product, TPware adapters are of two general types:

- Input language adapters
- Output communications adapters

The following sections describe their behavior.

1.5.1 Input Language Adapters

Input adapters present the components of an interface to clients that are written in a particular language such as C or that use a particular calling mechanism such as Automation. The TPware input adapters present to a client the following aspects of the TPware interface:

- A way to call each procedure within the interface

  TPware interfaces such as STDL task groups consist of one or more procedures that each take one or more arguments. These arguments may be input, output, or inout.

- A way to retrieve the exception information that may be returned on a TPware procedure call

  The TPware or adapter runtime software can generate exceptions. The called procedure can return exception information that is made accessible to the client through the STDL **einfo structure** (see Section 6.5).

- A way to specify the TPware call attributes for a call

  Clients must have a way to pass information such as destination and authentication information to communications adapters on each call. TPware does this through the use of the **call attributes string** (see Section 2.3).

1.5.2 Output Communications Adapters

Each output adapter handles specific call attributes and accepts a default for each attribute (see Section 2.3). These output adapters perform the appropriate transformation of the call into the communications messages required to invoke the remote task.

1.6 HP TP Desktop Connector Interfaces to TP Applications

The *HP TP Desktop Connector* product provides the following interfaces for client access to TP applications (illustrated in Figure 1–4):

- Client Services Interface (described in other manuals; see "Associated Documents" in the Preface)

- TPware Adapter Interface (described in this manual)

Figure 1–4   HP TP Desktop Connector Interfaces
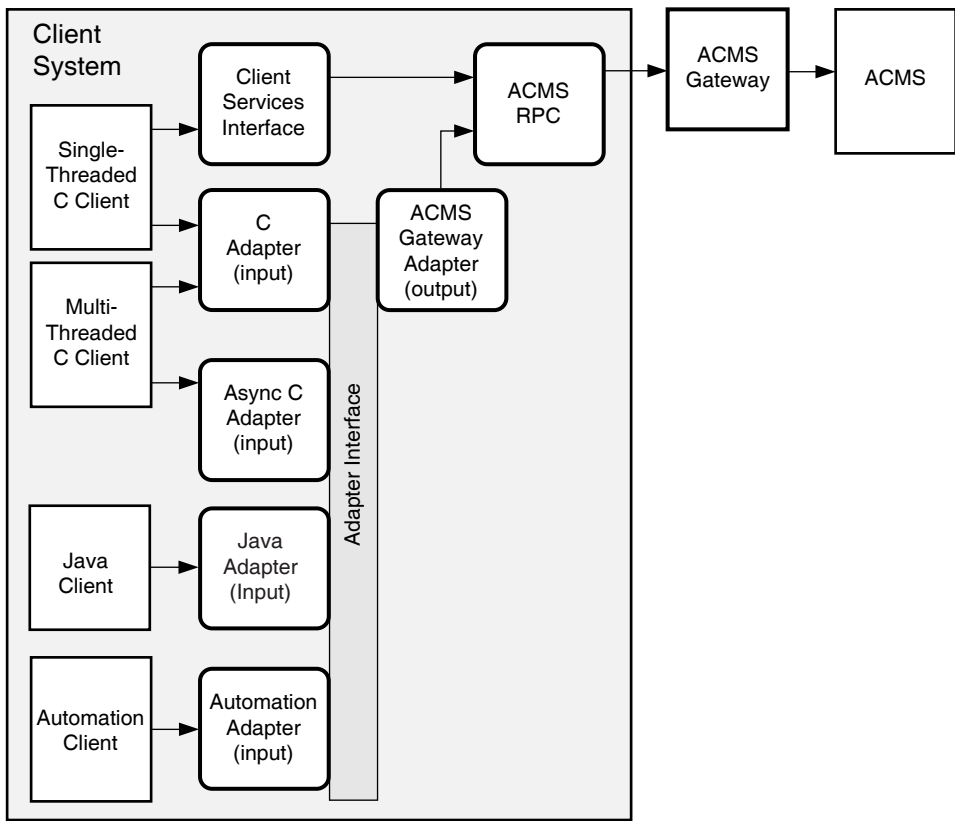
VM-1053A-AI

The **client services interface** provides single-threaded client access to ACMS applications. The client makes a series of service calls to Client Services to invoke tasks and handle exchange I/O from the tasks.

Figure 1–4 shows how a single-threaded C client calls Client Services, which uses ACMS RPC to call the HP TP Desktop Connector gateway. This ACMS gateway executes on OpenVMS platforms.

The **TPware adapter interface** provides client access to ACMS using adapters generated by the TPware STDL compiler. To call ACMS tasks, use the ACMSADU compiler to generate an STDL task group definition for an ACMS application (see Section 1.7), and then use the STDL compiler to generate the adapter.

The type of client determines the type of input adapter:

- C input language adapter

  Provides both single-threaded and multithreaded C client program access to ACMS.

- Asynchronous C input language adapter

  Provides an asynchronous interface for C clients to call ACMS applications. When the task call completes, the adapter invokes the call-back routine provided by the client.

- Automation input language adapter

  Provides an Automation interface to ACMS applications. Use the Automation input adapter with any client that can use an Automation interface (currently limited to Windows platforms). For example, you can use tools such as Visual Basic and Office 2000.

- Java/JavaBeans input language adapter

  Provides a Java interface to ACMS applications. To the application Java client, the Java input adapter presents Java classes that represent application objects. It provides to those Java classes a Java Native Interface (JNI) that represents equivalent application objects. The Java classes call into the native methods, which in turn call the standard C interface.

The type of output adapter depends on the TP application being accessed and the communications mechanism being used. For example, the ACMS Gateway Adapter uses the ACMS RPC to send the call information to the gateway. The gateway process in turn calls the ACMS task.

1.7 The ACMSADU Extension

To invoke an ACMS task using a TPware adapter, you need to generate an STDL definition for the ACMS application. You do this by using an extension to the ACMSADU compiler, as shown in Figure 1–5.

Figure 1–5   Using the ACMSADU Extension

ML014393
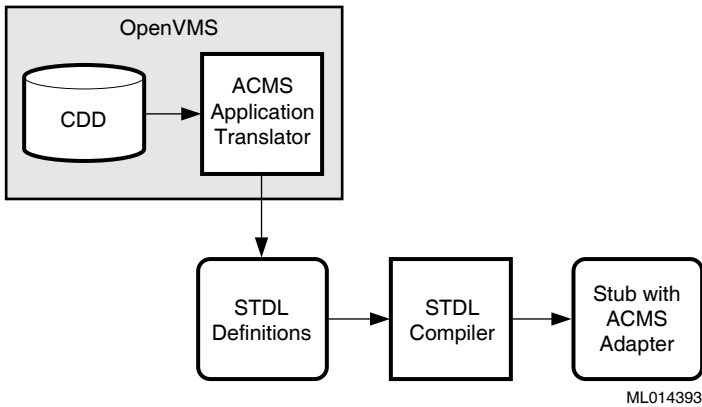
The ACMS application translator (the ACMSADU compiler extension) generates an STDL task group specification but applies the ACMS application name as the task group name. The STDL task name is mapped to an ACMS task name within the ACMS application.

After using the ACMSADU compiler extension on an OpenVMS system to generate the STDL task group, you copy the task group specification to a Windows system and use the STDL compiler there to generate the stub containing the ACMS Gateway adapter. For more detail on using the ACMSADU Extension, see Chapter 3.

2

Overview of Building a Client Interface

This chapter discusses the general procedure for building a client interface to your current application. Also see the online instructions for building samples supplied with the HP TP Desktop Connector product.

2.1 General Steps in Building a Client Interface

The HP TP Desktop Connector product builds a client interface to an existing TP application executing under ACMS software. To build a client, follow these general steps:

1. On the server development system, find the source code for the existing TP application.

   - If you are developing a client program to access existing ACMS tasks, do the following:

     a. Using the ACMSADU extension installed on the ACMS server development system, process the TDL source files to generate a source file that contains an STDL task group specification and data type definitions for the ACMS tasks. See Chapter 3 for instructions on preparing TDL applications.

     b. Copy the generated source file from the OpenVMS system to the client development system.

     c. Edit the generated STDL task group specification to supply a valid UUID (see Section 2.2).

2. On the client development system to which you copied the STDL source code, establish the environment for using the STDL compiler (see Section 4.1).

3. On the client development system, use the STDL compiler to process the STDL source files and to produce the output that facilitates client program development (see Section 4.2).

_____ **Note** _____

When building Automation or C clients, you can perform this STDL
compile step and, for Automation clients, the subsequent build step
using the client build utility GUI (see Chapter 5).
_____

4. Build the client interface.

   See the following chapters that explain the specific steps for using the
   STDL compiler, for building the client executable, and for setting up
   management for different types of clients:

   • Chapter 6 for writing and building C and asynchronous clients

   • Chapter 7 for writing Automation clients

   • Chapter 8 for writing Java clients

5. On a Windows client execution system, set up the management
   environment for the client interface (see Chapter 9).

2.2 Generating a UUID

If you are developing a client that calls an ACMS task, replace the default zero
UUID in your generated STDL task group specification with a valid UUID.

To generate a UUID use the `guidgen` application.

1. Invoke `guidgen` from either a DOS command line or the `Tools` menu
   item in Microsoft Visual C++. See the HP TP Desktop Connector SPD for
   supported versions of Microsoft Visual C++.

2. Select the `Registry Format` menu and `Copy` command to copy the UUID to
   the Windows clipboard.

3. Paste the UUID into the generated STDL task group specification.

4. After you paste the UUID, replace the braces, ({) and (}), with double
   quotes (").

2.3 Specifying Call Attributes and Management Information

When your client calls ACMS tasks , it can pass information such as destination and authentication to output adapters on each call. HP TP Desktop Connector software does this through the use of the **call attributes**. A call attribute has the following syntax:

```
name:value [ :value ] [, ... ]
```

If the client specifies a call attribute value without specifying a name for it, then a default name is assumed. Section 2.3.1 describes the call attributes and default value for each adapter that supports passing call attributes.

If you omit a call attribute or specify a null call attributes string, the management GUI possibly provides the attribute information (see Section 9.2).

2.3.1 Call Attributes for Calling ACMS Tasks

The ACMS Gateway adapter uses call attributes from either the client or from management information. The client program can specify the call attributes as a string using the following syntax:

```
application[:name],node[:name],authorization[:account-name:password]
```

Table 2–1 describes the call attributes and values.

Table 2–1 ACMS Gateway Adapter Call Attribute Values

Call Attribute Value	Description
Application[:*name*]	Name of the ACMS application (STDL task group name) being called
Node[:*name*]	Name of the TCP/IP node on which the ACMS gateway resides
Authorization[:	OpenVMS authorization information, requiring two values separated by a colon (:) character
account-name:	Name of the OpenVMS user account under which the task call is made
password]	Password for the OpenVMS user account under which the task call is made

By default, the application called is taken from the STDL task group name. The `application` attribute value accommodates the situation in which the client wants to call numerous ACMS applications that all contain the same number of tasks, with the same task names, and all take the same number of similarly defined arguments. The software assigns to the DLL the name under

which the task group is compiled, but overrides the application being called at runtime based on the value of this call attribute. Refer to Section 9.2.3 for further descriptions of these attributes. For example, a call attributes string for a client program might be:

```
"application:add_acms_appl,node:bigwig.corp.com,authentication:jones:sam"
```

------- **Note** -------

Do not use spaces in the string.

In the example, the `application` attribute has the value `add_acms_appl`, the `node` attribute has the value `bigwig.corp.com`, and the `authentication` attribute has the value `jones:sam` (the username and password to use for the call).

If you omit a call attribute or specify a null call attributes string, the management GUI provides the attribute information.

2.4 Exception Information

EINFO is the STDL mechanism for storing exception information generated during a call.

2.4.1 The STDL EINFO Definition

The coding for the STDL EINFO definition is shown in Example 2–1.

Example 2–1   EINFO Data Type Definition

```
TYPE EINFO IS
 RECORD
   ECLASS  INTEGER;
   ECODE   INTEGER;
   EPROC   TEXT CHARACTER SET SIMPLE-LATIN SIZE 32;
   EPGROUP TEXT CHARACTER SET SIMPLE-LATIN SIZE 32;
   ESOURCE INTEGER;
   ECGROUP UUID;
 END RECORD;
```

When an exception is returned to the caller, the following information is available:

- Exception class
- Exception code and exception code group

- Exception level
- Exception source
- Exception location

Sections 2.4.1.1 to 2.4.1.5 describe the information. Section 2.4.2 describes how a caller examines the returned exception information.

2.4.1.1 Exception Class

An exception class is a grouping of exception conditions that is based on the type of recovery action that can be taken and that allows for portable exception handling. The exception class identifies the exception type as a nontransaction exception or as a transaction exception. Each exception class is initially raised as a specific exception type. The exception type can change depending on how the exception is reported.

2.4.1.2 Exception Code and Exception Code Group

An exception code is a detailed classification of an exception condition. For application-generated exceptions, the exception code is portable and is defined in a message group definition associated with the STDL task group.

2.4.1.3 Exception Level

An exception level is a classification of exception conditions based on whether the exception was generated by the execution of a called task that propagated the exception to the calling client.

An exception level is defined as either current or propagated. An exception level is defined as **current** when the current task execution generates the exception and when an exception is encountered during the invocation of a task.

An exception level is defined as **propagated** when the execution of a task called by the client interface generates the exception. A client cannot distinguish between a current exception and a propagated exception.

2.4.1.4 Exception Source

An exception source is a classification of exception conditions based on whether the exception was raised by the application or by the TPware runtime system. A value of zero indicates that the exception was raised by the system. A value of one indicates that the exception was raised by the application.

2.4.1.5 Exception Location

The exception location consists of the following text fields in EXCEPTION-INFO-WORKSPACE that contain names describing where the exception occurred:

- The name of the task (if the task raised the exception) in which the exception occurred, or an arbitrary string filled in by the application

- The name of the task group (if the task raised the exception) in which the exception occurred, or an arbitrary string filled in by the application

When an exception is raised in a task, the system sets the exception procedure to the task name, and the exception procedure group to the task group name.

2.4.2 Examining Returned Exception Information

Client programs examine the language-specific implementation of the STDL EINFO information upon completion of the call to the task. The TPware runtime system returns an exception by setting the STDL-defined external variable with agreed-upon standard values. The client program examines the exception data from the EINFO record returned as a result of the task call to determine whether an exception is raised.

1. If the ECLASS field contains a value of zero, no exception is returned and output arguments are returned.

2. If the ECLASS field contains a nonzero value, the following conditions apply:

   - An exception is returned. The ECODE field contains a nonzero value that indicates which exception occurred.

   - Output arguments are undefined.

3. If an exception is returned, the client program can examine the ESOURCE field to determine whether the exception was generated by the system (exception source is 0) or by the application (exception source is 1).

HP TP Desktop Connector provides an implementation-specific means to check these fields. For example, if the client is written in C, the einfo.h file defines the C structure einfo that the program can access to check status. When an exception is returned to the C program, the EINFO information is available in members of the einfo structure. The fields in the EINFO data type definition correspond to members of the einfo structure in C.

Sections 2.4.2.1 and 2.4.2.2 explain the possibilities for error codes in the ECODE field.

2.4.2.1 TPware Error Codes

If the `ECODE` field contains a value from 01 to 255 (0x01 to 0xFF), this signals a TPware runtime exception or data conversion errors in the output adapter. The TPware runtime exceptions and errors are described in the file `stdlrt_msg.h`.

The file is located in a directory with a name in the following format:

install-directory\stdl\include

For example, if you installed the product in the `C:\tpware` directory, then the STDL message file is located at:

`C:\tpware\stdl\include\stdlrt_msg.h`

If the client calls an ACMS task (uses the ACMS Gateway adapter), the `ECODE` field can alternatively contain an ACMS client error code (see Section 2.4.2.2).

The software provides language-dependent code that returns the error message text translated from the value in the `ECODE` field, as follows:

- For the C language clients, see Section 6.2.
- For Automation clients, see Section 7.5.
- For Java clients, see Section 8.3.1.

2.4.2.2 ACMS Client Error Codes

If the client calls an ACMS task (uses the ACMS Gateway adapter), the `ECODE` field can contain either a TPware error code (see Section 2.4.2.1) or a value from -3000 to -3199 (0xFFFFF448 to 0xFFFFF381). The negative value signals an ACMS Gateway adapter runtime exception. If this exception is related to an ACMS error, text related to the error may be found in an argument passed back to the client in the task call (see Section 6.2.1.2).

These exceptions are described in the file `acmsda_client_messages.txt` located in a directory with a name in the following format:

install-directory\stdl\include

For example, if you installed the product in the `C:\tpware` directory, then the client message file is located at:

`C:\tpware\stdl\include\acmsda_client_messages.txt`

2.5 Taking the Next Step

If your client is accessing an ACMS application, go to Chapter 3 for a description of the steps necessary to prepare ACMS applications for client access.

Chapter 4 describes the STDL compiler. This compiler generates the TP stubs from the application's STDL task group specification files. You can use this compiler for all client types.

Chapter 5 describes the client build utility that you can use if you are building Automation or C clients. This build utility simplifies the processes used when using the STDL compiler.

Chapters 6 to 8 present specific procedures for different types of clients.

3

Preparing ACMS Applications for Client Access

If your client is going to call ACMS tasks, you need to perform some processing on the ACMS development system where the ACMS application is defined. This chapter explains how the HP TP Desktop Connector components on ACMS systems are used so that client programs, including web clients, can be built to access ACMS applications.

3.1 HP TP Desktop Connector Components

If your client calls ACMS tasks, use HP TP Desktop Connector components that you install and run on an ACMS system. The HP TP Desktop Connector software includes the following components that support the generation and use of client programs that access ACMS applications:

- HP ACMS Application Definition Utility (ACMSADU) extension

  Enables ACMSADU to translate an ACMS application into an STDL task group specification.

- A gateway

  Supports the ACMS RPC protocol between the ACMS Gateway output adapters on the calling system and ACMS applications on the called system.

The HP TP Desktop Connector components on the OpenVMS system (the ACMSADU extension and the gateway) reside on systems running ACMS software. The components can be on the same ACMS system or on different ACMS systems (refer to *HP TP Web Connector Installation Guide*).

3.1.1 ACMSADU Extensions

The extensions in the modified ACMSADU image enable you to translate an ACMS application to an STDL task group specification. HP TP Desktop Connector extension commands allow you to control the translation. ACMSADU performs the following HP TP Desktop Connector functions:

- Extracts information about ACMS tasks and records.

- Translates the extracted information to STDL format.

- Writes the translated information in a form that the STDL compiler can read.

Figure 3–1 shows how ACMSADU reads the application, group, task, and record definitions from the Common Data Dictionary (CDD).

Figure 3–1 ACMSADU Translation Model

ML014076
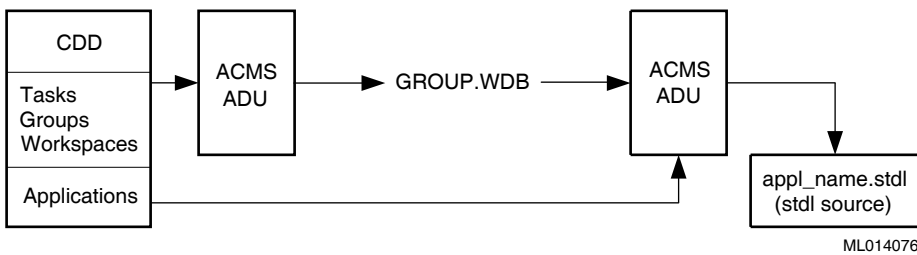
ACMSADU produces as output an intermediate group file, from which it produces the STDL task group specification, including record definitions and task group headers with records as arguments.

3.1.2 HP TP Desktop Connector Gateway for ACMS

The gateway runs on an ACMS system and connects the ACMS Gateway adapter on the client system to the ACMS system on which the target application is running. You provide information that enables the connection between your client program and the ACMS tasks being called (see Section 9.2). Use command procedures provided with the software to control and manage the gateway (see *HP TP Desktop Connector for ACMS Gateway Management Guide*).

3.2 Running the ACMSADU Extension

Translating applications from TDL to STDL performs the following operations:

- Group_task translation (see Section 3.2.1)

- Application_group translation (see Section 3.2.3)

- Record definition translation and conversion (see Section 3.2.7)

You perform these operations with HP TP Desktop Connector qualifiers for the ACMSADU BUILD GROUP and ACMSADU BUILD APPLICATION commands. Refer to *HP ACMS for OpenVMS* ADU Reference Manual for a listing of qualifiers.

Section 3.2.2 describes the actions that occur during group_task translation, and Section 3.2.4 describes the actions that occur during application_group translation.

3.2.1 Group_task Translation

Group_task translation produces temporary files containing ACMS task and task group information. Group_task translation must be performed for every ACMS group that comprises an application.

Synopsis

The command syntax is:

BUILD GROUP *acms_group_name* [/STDL]

Description

The *acms_group_name* parameter refers to the name of the ACMS group.

The /STDL qualifier directs ACMSADU to output task and record information that is used for the final translation to STDL. The output file produced by /STDL has a name in the following format:

GROUP.WDB

The .WDB file type indicates the intermediate-format web database file. The BUILD GROUP command writes the file to the default directory (refer to *HP ACMS for OpenVMS* ADU Reference Manual for information on DEFAULT TASK GROUP FILE).

The /NOSTDL qualifier instructs ACMSADU not to produce the intermediate-format file. The default parameter is /NOSTDL.

3.2.2 Actions During Group_task Translation

During group_task translation, ACMSADU performs the following actions:

- Reads information from the CDD.

- Processes the read information.

- Creates the intermediate-format file.

ACMSADU processes the group description and the description of any tasks and records associated with that group. ACMSADU builds descriptions of all records according to the CDD information. Then, ACMSADU creates a description of all tasks in the group along with the names of records used as parameters to those tasks. The intermediate-format file in which ACMSADU writes all this information is used as input to the application_group translation (see Section 3.2.3).

3.2.3 Application_group Translation

Application_group translation reads the group names specified in the application from the CDD and translates the intermediate-format group file built from the group_task translation to its equivalent STDL task group specification (see Section 3.2.1).

Synopsis

The command syntax is:

BUILD APPLICATION *application_name* [/STDL]

Description

The parameter *application_name* refers to the name of an ACMS application.

The /STDL qualifier directs the translator during the ACMSADU BUILD APPLICATION to generate STDL code using the intermediate-format group file that was created during the ACMSADU BUILD GROUP compilation. The output is a file with a name in the following format:

application_name.STDL

―――――――――――――――――― **Note** ――――――――――――――――――

As indicated by this format, ACMSADU creates an STDL task group specification but applies the ACMS application name as the STDL task group name.

―――

ACMSADU generates in the default directory an STDL source file containing a task group specification and related data type definitions. The output file name *application_name* is derived from the ACMS application name. You use this file to create the client interface (see Section 3.2.5).

3.2.4 Actions During Application_group Translation

ACMSADU accesses the CDD to find the group name and reads application information from the CDD including the ACMS group name. Then ACMSADU uses the application name to locate and to read the corresponding group temporary file. Next, ACMSADU processes the internal descriptions of the records and writes them in STDL format with duplicate names removed (see Section 3.2.6 for the restrictions).

After each task name is processed, ACMSADU generates the STDL TASK ARGUMENT statement, which includes the task parameters and an indication as to whether they are input or output. The arguments are as follows:

- A record containing a text field of 256 characters for the selection string as an input argument.

- A record containing a text field of 80 characters for the extended status as an output argument.

- Any arguments defined in the TDL task definition, in order.

The resulting file that ACMSADU generates contains an STDL task group specification that you use in the next stage of development (see Section 3.2.5). See Section 3.2.6 for information on ACMSADU restrictions.

3.2.5 Using the Output of Application_group Translation

Use the file containing the STDL task group specification that ACMSADU generates (see Section 3.2.3) to generate the adapters for the client. The next steps you take are on the client development platform, as follows:

1. Copy the STDL task group specification to the client development system on which you have HP TP Desktop Connector installed and to the directory in which you are going to build the client.

2. Edit the STDL task group specification on the client development system to provide a nonzero UUID (see Section 2.2).

3. Follow the procedures for the type of client, as follows:

   - For the C client, see Chapter 6.

   - For the Automation server, see Chapter 7.

   - For the Java client, see Chapter 8.

3.2.6 ACMSADU Extension Restrictions

ACMSADU has the following restrictions:

- One ACMS task group per application

  Each application is limited to exactly one ACMS task group.

- An ACMS task cannot be renamed in the ACMS application definition.

- Exchange I/O not supported

  Exchange I/O statements are not supported. Refer to the **HP ACMS for OpenVMS** *ADU Reference Manual* for information on building no I/O tasks.

- COLUMN MAJOR feature not supported

  HP TP Desktop Connector software does not allow the reordering of rows and columns of an array to support the COLUMN MAJOR feature of the CDD.

- Record names must be unique

  All record names within a group must resolve to a unique name. STDL record names are the CDD simple names. If two CDD record names have the same simple name, edit the source code to differentiate the two records.

- All tasks are processed as GLOBAL

  ACMS software allows tasks to be identified as LOCAL or GLOBAL in the task, group, and application definitions. For the purposes of STDL translation, all tasks are processed as GLOBAL tasks. Therefore, when you build an application with the /STDL qualifier, all tasks appear in the .STDL file that the ACMSADU utility creates. If a HP TP Desktop Connector client calls a LOCAL ACMS task, the runtime system flags the call as an error.

3.2.7 Converting Records

ACMS software uses records to allow data to be passed from user agents or forms acting as agents, through the ACMS system to ACMS servers, and back to the caller. These records are buffers containing record-like structures made up of fields. Each field has a name, a data type, and other attributes.

The data descriptions used by ACMS software to describe records are defined by either the CDDL utility in DMU format or the CDO utility in CDO format, and then are placed into the CDD. Record descriptions are extracted from the CDD by various OpenVMS and user programs, including agents, forms systems, the ACMS system, and OpenVMS compilers when compiling ACMS servers. When creating an STDL task group specification from an ACMS

application, ACMSADU extracts record descriptions from the CDD and translates them to STDL format.

3.2.8 Translating Data Types from OpenVMS to STDL

Fields within the ACMS records all have an OpenVMS data type. ACMSADU creates the #Pragma statements in the generated data type definitions. During compiling, ACMSADU selects a corresponding STDL data type for each OpenVMS data type that can be expressed in the CDD. Not all OpenVMS data types are represented in STDL, so the ACMS Gateway adapter makes some conversions at runtime according to the ACMSADU selections. Tables 3–1 to 3–4 summarize the OpenVMS data type support for the HP TP Web Connector Gateway for ACMS software. Those data types labeled NOT SUPPORTED are flagged as warnings by the STDL compiler.

Table 3–1 describes integer support.

Table 3–1 Integer Support for the ACMS Gateway Adapter

OpenVMS Data Type	STDL Data Type	OpenVMS Pragma Data Type
SIGNED BYTE	INTEGER SIZE 1	None
UNSIGNED BYTE	UNSIGNED INTEGER SIZE 1	None
SIGNED WORD	INTEGER SIZE 2	None
UNSIGNED WORD	UNSIGNED INTEGER SIZE 2	None
SIGNED LONGWORD	INTEGER SIZE 4	None
UNSIGNED LONGWORD	UNSIGNED INTEGER SIZE 4	None
SIGNED QUADWORD	ARRAY SIZE 8 OF OCTET	None
UNSIGNED QUADWORD	ARRAY SIZE 8 OF OCTET	None
SIGNED OCTAWORD	ARRAY SIZE 16 OF OCTET	None
UNSIGNED OCTAWORD	ARRAY SIZE 16 OF OCTET	None

Table 3–2 describes floating point and complex data type support.

Table 3–2 Floating Point and Complex Data Type Support for the ACMS Gateway Adapter

OpenVMS Data Type	STDL Data Type	OpenVMS Pragma Data Type
F_FLOATING	FLOAT SIZE 4	F_FLOATING
D_FLOATING	FLOAT SIZE 8	D_FLOATING
G_FLOATING	FLOAT SIZE 8	G_FLOATING
H_FLOATING	ARRAY SIZE 16 OF OCTET	None
S_FLOATING	FLOAT SIZE 4	None
T_FLOATING	FLOAT SIZE 8	None
F_FLOATING COMPLEX	RECORD R FLOAT SIZE 4; I FLOAT SIZE 4; END RECORD;	F_FLOATING for each field
D_FLOATING COMPLEX	RECORD R FLOAT SIZE 8; I FLOAT SIZE 8; END RECORD;	D_FLOATING for each field
G_FLOATING COMPLEX	RECORD R FLOAT SIZE 8; I FLOAT SIZE 8; END RECORD;	G_FLOATING for each field
H_FLOATING COMPLEX	ARRAY SIZE 32 OF OCTET	None

Table 3–3 describes decimal data type support.

Table 3–3 Decimal Data Type Support for the ACMS Gateway Adapter

OpenVMS Data Type	STDL Data Type[1]	OpenVMS Pragma Data Type
PACKED DECIMAL	DECIMAL STRING SIZE X	PACKED DECIMAL
UNSIGNED NUMERIC	DECIMAL STRING SIZE X	UNSIGNED NUMERIC
LEFT OVERPUNCH NUMERIC	DECIMAL STRING SIZE X	LEFT OVERPUNCH NUMERIC
LEFT SEPARATE NUMERIC	DECIMAL STRING SIZE X	LEFT SEPARATE NUMERIC

[1]The maximum SIZE for the decimal data type is limited to 18.

Table 3–3 (Cont.) Decimal Data Type Support for the ACMS Gateway Adapter

OpenVMS Data Type	STDL Data Type[1]	OpenVMS Pragma Data Type
RIGHT OVERPUNCH NUMERIC	DECIMAL STRING SIZE X	RIGHT OVERPUNCH NUMERIC
RIGHT SEPARATE NUMERIC	DECIMAL STRING SIZE X	RIGHT SEPARATE NUMERIC
ZONED NUMERIC	DECIMAL STRING SIZE X	ZONED NUMERIC

[1]The maximum SIZE for the decimal data type is limited to 18.

Table 3–4 describes other data type support.

Table 3–4 Other Data Type Support for the ACMS Gateway Adapter

OpenVMS Data Type	STDL Data Type	OpenVMS Pragma Data Type
DATE	DATE	VMS DATE
VARYING STRING	Not supported	

3.2.9 Translation of Other Record and Field Properties

Other properties may be defined in the CDD for records and fields. ACMSADU supports translation of the following property to STDL:

OCCURS

The OCCURS field declares fixed-length, one-dimensional arrays.

```
OCCURS  n TIMES
```

4

Compiling STDL Applications

This chapter presents environment information and STDL command line
reference material for use with HP TP Desktop Connector software in
generating adapter stubs used for client access to TP applications.

_____ **Note** _____

For information about invoking the STDL compiler through the client
build utility graphical user interface (GUI) on a Windows system, see
Chapter 5.

_____

4.1 STDL Compiler Environment Setup

To run the STDL compiler, you must register the environment variables needed
by the compiler. From a command window, run the `stdl_set_version.bat`
batch file from the directory named in the following format:

install-directory\stdl\bin

The value *install-directory* is the drive and directory in which you installed the
HP TP Desktop Connector software. For example, if you installed the product
in the `C:\tpware` directory, you would type the following:

`C:\tpware\stdl\bin\stdl_set_version.bat`

When execution completes, the STDL development environment is set up.

4.2 Using the STDL Compiler

The `stdl` command invokes the STDL compiler. The STDL compiler translates
STDL definitions and specifications contained in the input source file and
produces both object code to link with the clients and other outputs to aid
source code development. For the HP TP Desktop Connector product, the
STDL compiler generates adapter stubs used for client access to STDL tasks
and ACMS applications.

Synopsis

The `stdl` command syntax is:

```
stdl [-cswLMNX]

     [-a in_adapter:out_adapter]

     [-h directory] [-i path]

     [-o directory] [-S include]

     [-u uuid] [-v version] source-file-name
```

source-file-name

The input STDL source file designated by *source-file-name* must contain a task group specification. Use a nonqualified or fully qualified file name as input to the `stdl` command. All files specified with nonqualified names must be in the current working directory. The compiler appends the `.stdl` suffix to the name of the STDL input source file if you omit a suffix in the source file name.

Option Flags

-a *in_adapter*:*out_adapter*

Specifies an input adapter and an output adapter to provide interfaces between a calling client and a task.

Input adapters are described in Table 4–1.

Table 4–1 HP TP Desktop Connector Input Adapters

Adapter	Description
async	Provides an asynchronous interface for a C-language client. To start the task call, the client calls an adapter procedure whose function prototype is generated by the STDL compiler. The call returns before the task call completes. When the task completes, the adapter invokes the callback routine provided by the client (see Section 6.4).
auto	Provides an interface for a client developed in an Automation language (see Section 7.3).
c	Provides a synchronous interface for a C-language client (see Section 6.2).

Table 4–1 (Cont.)   HP TP Desktop Connector Input Adapters

Adapter	Description
java	Provides an interface for a Java client. The Java classes that represent the application objects present their data record fields as public members (see Section 8.3).
javabeans	Provides an interface similar to JavaBeans for a Java client. The Java classes that represent the application objects present their data record fields through accessor methods (see Section 8.3).

Output adapters are described in Table 4–2.

Table 4–2   HP TP Desktop Connector Output Adapters

Adapter	Description
acmsda	Produces an interface that calls ACMS tasks using ACMS RPC communications.

The flag produces an adapter stub that contains the particular input adapter suited to the language in which you developed the client and a specific output adapter designed to call the task. Based on the input and output adapters specified, other output files are produced (see the section titled "Compilation Output" later in this chapter).

-c

Generates a C header file (see the section titled "Compilation Output" later in this chapter). Use this option for clients that use a C or asynchronous C input adapter. The file is written either into the current working directory or into the directory specified by the –h flag.

-h *directory*

Specifies a directory into which the STDL compiler generates C header files. By default, the compiler generates the C header files in the current working directory.

-i *path*

Specifies a directory search path for STDL #INCLUDE and #CINCLUDE files. A *path* argument must be either a legal directory specification or an environment variable. If you specify multiple arguments, separate each by a semicolon (;), and enclose environment variables in percent characters (%).

For example:

```
/a/b;/b/b;%PATH%
C:a\a;D:b\b;%PATH%
```

The compiler searches for the #INCLUDE or #CINCLUDE files in the following order:

- Path containing the top-level source file

- Path specified by the `-i` flag

- Directories listed in the STDL_INCLUDE_PATH environment variable

-L

Produces a line-numbered source code listing file (see the section titled "Compilation Output" later in this chapter). Use the `-S include` flag with this flag to include in the listing file the contents of the files specified in the #INCLUDE and #CINCLUDE directives.

-M

Specifies that the STDL source files use Multivendor Integration Architecture (MIA) syntax. The default is X/Open syntax.

-N

Terminates with a null character each string contained in workspaces in the STDL source files. The default is to pad with space characters unused storage in strings in workspaces.

-o *directory*

Specifies a drive and directory into which the compiler generates output files (see the section titled "Compilation Output" later in this chapter). By default, the compiler generates these output files in the current working directory.

-s

Performs only syntax and semantic checks on the input source file. Does not generate any output files. By default, the compiler generates output as directed by other options.

-S include

When used with the `-L` flag, includes in the line-numbered source code listing file the contents of the files in the #INCLUDE and #CINCLUDE directives. If you specify the `-L` flag without the `-S include` flag, the compiler includes only the input source file in the listing file.

-u *uuid*

Specifies the default universal unique identifier (UUID) for the first processing group specification in the input file that does not have a UUID specified in the STDL syntax. If the input file contains a processing group specification with no UUID specified, this flag is required. The –u flag applies only to processing groups that you are compiling, not to processing groups to which the STDL source refers. A UUID value comprises hexadecimal digits (X) grouped as follows:

`"XXXXXXXX-XXXX-XXXX-XXXXXXXXXXXX"`

-v *version*

Specifies the default version number for any task group or processing group specification in the input file that does not designate a version in the specification syntax. The compiler defaults to assigning a version of 0.0 for any processing group with no syntax-specified version unless you specify a version with the –v flag. For *version*, specify a decimal literal without a sign. Processing group specifications included for reference only do not require a version number on the command line.

-w

Suppresses warning messages. The default is to display warning messages.

-X

Specifies that the STDL source files contain X/Open syntax.

The STDL compiler defaults to X/Open STDL syntax. To compile Multivendor Integration Architecture (MIA) syntax, use the –M flag.

Description

The STDL compiler generates adapter stubs used for client access to ACMS tasks. (Task compilation is not supported.) The compiler directs output to the current working directory, the directory containing the top-level input source files, or to directories specified by the option flags or by the values set by environment variables.

The compiler converts the names of the task groups as follows:

- Changes the letters to lowercase
- Replaces any hyphen characters (-) with underscore characters (_).

Compilation Output

The STDL compiler generates the following output files based on the contents of the STDL source files and on the flags used during compilation:

- Listing file with a name in the following format:

  source-file-name.lis

  The value *source-file-name* is the same name as the input source file and has a suffix of `.lis` that signifies a listing file.

  This file contains line-numbered source code statements, with error messages inserted directly under each source line containing an error, and a summary report. The file is generated when you use the `-L` flag.

- C header file with a name in the following format:

  source-file-name.h

  The value *source-file* is same name as the input source file and has a file type of `.h` to designate a C header file. The header file contains function prototypes and C-language definitions for STDL data types. This file is generated when you use the `-c` flag but omit the `-s` flag.

  Include this file in any C programs that call ACMS tasks, or that refer to STDL data type definitions.

- Adapter stub

  The name is in the following format:

  group_in_adapter_out_adapter.obj

  This file uses the converted name of the task group (*group*) and contains the adapter stub created with the `-a` flag and with the adapters specified in the *in_adapter* and *out_adapter* arguments. (If you specify an asynchronous input adapter, c appears in the name, not `async`.)

  The compiler generates the adapter stub if you use the `-a` flag but not the `-s` flag.

  Link the adapter stub as follows:

  – With the C client executable if the input adapter is `c` or `async` (see Chapter 6)

  – With the resource file *group*`_auto.res` if the input adapter is `auto` (to create an Automation server DLL, see Section 7.1)

- Resource file with a name in the following format:

  *group*_auto.res

This file uses the name of the task group *group* and provides access to a type library for use with an Automation adapter. The compiler generates this file if you specify the -a flag with an auto input adapter.

To create an Automation server DLL, link this OLE resource object with an adapter stub that includes an Automation adapter, with a name in the following format:

group_auto_out_adapter.obj

See Section 7.1.

- Java archive file with a name in the following format:

group_out_adapter.jar

This standalone file contains the Java classes for the task group with *group* as the converted name of the task group. The compiler generates this file if you specify the -a flag with java or javabeans as the input adapter. The file is used at runtime.

5

Using the Client Build Utility

This chapter describes how to use the TPware client build utility and its GUI to build Automation or C clients on client systems.

5.1 Building Client Programs

The TPware client build utility provides a GUI that makes it easier for you to create Automation or C clients. You provide input through the GUI, and the utility creates the files needed to build the client program.

As input, you provide the utility with an STDL task group file name, an input adapter type, an output adapter type, and an output directory name. The utility places the created files in the specified output directory along with a log file that describes the results of running the utility, including errors if any occurred. The utility notifies you whether the build succeeded or failed.

Although the client build utility eliminates some of the complexity of having to create a makefile (the utility creates the client makefile), you need to refer to Chapter 2 and the specific client chapters for other steps associated with creating a client.

Using the client build utility requires that you fulfill the following prerequisites:

- Your client development must be on a supported client platform. See the HP TP Desktop Connector SPD for supported client platforms.

- Visual Studio must be in the path.

- Your STDL task group specification must contain a valid UUID. If you use ACMSADU to create your task group, generate the UUID and insert it into the task group's STDL code before using the client build utility (see Section 2.2).

- You must copy your STDL task group specification file and any related files (such as a file containing data type definitions) from your server system to your client system.

- Your STDL task group specification must contain only one task group.

The following sections describe some differences in the way the utility builds specific clients.

5.1.1 Building an Automation Client

If the input adapter type that you specify is Automation, the utility creates a makefile that invokes the STDL compiler and the platform linker to create a DLL that is invoked by the Automation client. The utility performs the following operations:

1. Invokes the STDL compiler to create the required object files in a temporary subdirectory.

2. Invokes the linker to link all object files from this subdirectory into the DLL.

3. Places the DLL in the output directory.

See Chapter 7 for information about registering this DLL and other steps for creating an Automation client.

5.1.2 Building a C Client

If the input adapter type that you specify is C, the utility creates a makefile that invokes the STDL compiler to create build files in the output directory. As shown in Table 5–1, the files generated for building a client depend on the output adapter type that you specify.

Table 5–1   Files Generated for a Client Build

Adapter Type	File Type	File Name
ACMS	Header	*input_file_name*.h
	Object	*task_group_name*_c_acmsda.obj

A header file (.h) is a file that you include in your C compile. Object files (.obj) are files that you link with your C client.

See Chapter 6 for more information about linking object files and libraries with your C client.

5.2 The Graphical User Interface

The client build utility allows you to easily enter your requisite build information through a GUI. By using the GUI, you do not need to use the STDL command-line interface and, in the case of Automation clients, you do not need to write a client build makefile.

To invoke the client build utility, choose TPware Client Build Utility from the TPware program group. When the TPware Client Build Utility screen opens, enter the information that is appropriate for the client that you want to build:

1. Enter the STDL Filename of your STDL task group specification, or click the Browse button to select the file name.

2. Enter an Output Directory name, or click the Browse button to select a directory. The utility generates output in this directory. If you do not specify an output directory, the utility generates the output files in the same directory that contains the input (that is, the same directory as the STDL task group specification file).

3. From the Input Adapter Type drop-down list, select one of the following:

   • C

   • Automation

4. From the Output Adapter Type drop-down list, select the output adapter type from a list that might include any of the following (depending on the TPware products that you have installed):

   • ACMS

5. Click the OK button. When the build is completed, the utility displays a message that notifies you whether the build was successful and gives you the option to view the log file.

In the output directory, the utility creates a subdirectory for temporary files. When the build is completed successfully, the utility deletes this subdirectory.

The utility also creates a log file, makefile, and a batch file in the output directory. If the object files or the DLL are created successfully, the utility deletes the makefile and batch file. However, if errors occur, the utility retains these files for debugging purposes.

Before invoking the makefile that invokes the STDL compiler, the utility runs the `stdl_set_version.bat` batch file to set up the environment for the STDL compiler to run successfully. The location of `stdl_set_version.bat` depends on an environment variable (STDL_DEV_DIR) that is set during installation of the product (see Section 4.1). If the utility cannot find this file via the environment variable, an error occurs.

6

Writing and Building C and Asynchronous Clients

Client programs written in C can call:

- TDL tasks executing under an ACMS system

6.1 Steps for Writing and Building the Client

To build a client program written in C, do the following:

1. Obtain the HP-related client development components depending on the type of tasks being called and the type of platform on which the client and server are being developed.

    - Perform the following steps to obtain the components:

        a. Copy to the client development system a task group specification and any related files from an ACMS system (see Section 2.1).

        b. Compile the STDL task group specification.

Note

When building a C client program you can perform the compile step using the client build utility GUI (see Chapter 5) rather than the command line interface described here. However, this utility does not support asynchronous clients.

On the STDL compiler command line, use the ‑c flag and the ‑a flag. See Section 4.2 for the complete `stdl` command syntax. With the ‑a flag, supply the appropriate input adapter and output adapter from those shown in Table 6–1.

Table 6–1 C Client Adapter Specifications

	Specification	Purpose
Input Adapters	async	Executes asynchronous calls to ACMS tasks .
	c	Executes synchronous calls to ACMS tasks or.
Output Adapters	acmsda	Calls ACMS tasks.

For example:

```
stdl -c -a c:acmsda test_task_group
```

The command generates a C header file and produces a C input adapter and a acmsda output adapter in the generated adapter stub. The compilation produces files with names in the following format:

group.h

group_c_out_adapter.obj

The format conventions are:

group   Converted name of the compiled STDL task group specification. When developing client interfaces for ACMS applications, the *group* name prefix is the ACMS *application_name* prefix (see Section 3.2.3).

c   Value for either the C or asynchronous input adapter. If you specify an asynchronous input adapter, c appears in the name, not async.

out_adapter   One of the values for the specified output adapters listed in Table 6–1.

See the section titled "Compilation Output" in Section 4.2.

2. Write the client procedures.

    a. Use the header file to code the task calls (see Section 6.2).

    b. Decide whether to use the call attributes string (see Section 6.3).

    c. If the C client program that you are building executes asynchronous calls, see also Section 6.4.

    d. Check status. After the call to the task, access STDL status using the einfo structure (see Section 6.5).

3. Compile the client program.

4. Link the following items to produce the client image. This step is the same for C or asynchronous C clients.

- Objects and libraries that depend on the type of task that the C client calls and the output adapter used, as shown in Table 6–2.

Table 6–2 C Client Adapter-Dependent Link Input

If Client Calls...	Input File Name	Comment
ACMS tasks (uses the acmsda adapter)	*group*_c_acmsda.obj	The adapter stub for the ACMS application (STDL task group)
	stdl_acmsda.lib	ACMS Gateway adapter runtime link library

- Client program files
  - All of the object files
  - Any libraries required by the object files
- The TPware runtime library:

  stdl_rtm.lib

6.2 C-Language Support

The STDL compiler generates C header files to support client development. Include the following files in your C client:

- Group header file *source-file-name*.h—Contains the following source code:
  - Function prototypes that define the interface to the tasks and the structure definitions for arguments passed to the called tasks (see Sections 6.2.1 to 6.2.4). The STDL compiler generates the contents based on the tasks defined in the task group specification.
  - A generated function for specifying the call attributes string (see Section 6.3).
  - The stdl_srtl_translate_ecode function to return the error message text translated from the value in the ecode field (see Section 6.5.3).
- Exception information header file einfo.h—Contains the C structure definition for the einfo external variable (see Section 6.5.1).

- Exception class header file `eclass.h`—Equates the symbolic name (class identifier) for each exception class with its exception class value (see Section 6.5.2).

6.2.1 Function Prototypes and Argument Passing

The STDL compiler creates a function prototype that contains a C external task name function declaration for each noncomposable task in a task group specification. The function prototypes are written to the group header file and have the formats described in Sections 6.2.1.1 and 6.2.1.2.

6.2.1.1 Task Call Function Prototype Format

Unless the calls are to ACMS applications (see Section 6.2.1.2), generated function prototypes have the following format:

converted-task-name([*argument* [,...]])

Task names are converted to function names according to the rules for identifiers (see Section 6.2.4). Task function declarations do not have a return value, and all arguments are passed as pointers.

In the client program, call the C procedure for the tasks as normal procedure calls.

For each task argument, declare a variable using the STDL data type definition from the group header file.

6.2.1.2 ACMS Task Call Arguments

If the calls are to ACMS applications, functions have two extra arguments in the following format:

converted-task-name(*string,status*[,*argument* [,...]])

The extra arguments are:

string	A selection string, consisting of one STDL record containing one 256-character field of data type ISO-LATIN-1 text.
status	An extended status string, consisting of one STDL record containing one 80-character field of data type ISO-LATIN-1 text. This extended status is message text associated with an ACMS error returned from the ACMS application.

See Section 2.4.2.2.

6.2.2 STDL to C Data Type Mapping

The group header file contains C definitions corresponding to record data type definitions in the STDL source files. Table 6–3 maps the STDL data types to the C data types. The STDL compiler generates header files with these mappings.

Table 6–3 Mapping STDL Data Types to C Data Types

STDL	C
ARRAY SIZE *n* OF *type*	*type id*[*n*]
ARRAY SIZE *n* OF ARRAY SIZE *m* OF *type*	*type id*[*n*] [*m*]

ARRAY SIZE *n* TO *m*
   DEPENDING ON *number*
   OF *type* [1]
	struct *rec* {
	   long int *number* ;
	   type id[*n*] ;
	} ;

DECIMAL STRING SIZE *a* SCALE *b*	char *id*[*a*+1] [2]
INTEGER	long *id* (or equivalent signed long, long int or signed long int)
OCTET	unsigned char *id*

TYPE *rec* IS RECORD
   id IS *type* ; END ;
	struct *rec* {
	   type id; } ;

TEXT SIZE *n* CHARACTER SET
   ISO-LATIN-1
   ISO-LATIN-2
   SIMPLE-LATIN
	char *id*[*n*]

UUID	uuid_t *id* [3]

[1]The number of repetitions of a variable length array is mapped as a record *rec* containing the number of repetitions and the repeated array.

[2]The data transferred is a character string in ISO 6093:1985, signed NR1 format, consisting of a row of one sign (or blank character) and digits that do not include characters indicating a decimal point. The user program should convert the representation to a decimal number.

[3]The uuid_t value is a typedef for the DCE UUID.

Key to Variables

a, *b*, *m*, *n*—An appropriate decimal number
id, *number*, *rec*—An appropriate name
type—A valid STDL data type or a user-defined data type identifier

6.2.2.1 Additional ACMS to C Data Type Support

If you are developing a client to call an ACMS application, additional data types are supported. The ACMSADU provides compilation support for some OpenVMS data types and the ACMS Gateway adapter provides runtime support (see Section 3.2.8). Data types in addition to those supported by STDL are supported for writing clients for ACMS applications as shown in Table 6–4.

Table 6–4 Mapping of Additional ACMS Data Types to C Data Types

STDL	C
DATE	char[17] where the bytes are in the following format:
	YYYYMMDDHHmmSSttt
	For example, New Year 2002 at midnight is "20020101000000000".
FLOAT SIZE 4	float
FLOAT SIZE 8	double
INTEGER SIZE 1	char
INTEGER SIZE 2	short
INTEGER SIZE 4	int
UNSIGNED INTEGER SIZE 1	unsigned char
UNSIGNED INTEGER SIZE 2	unsigned short
UNSIGNED INTEGER SIZE 4	unsigned int

6.2.3 Structures for STDL Data Types

C definitions of STDL data types have the `struct` format shown in Example 6–1.

The data type identifier from the STDL data type definition is used as the name of the C structure tag. Each field definition within the STDL record is a named member of the C structure. The order and number of the members of the structure within the C include file match the order and number of the field definitions in the STDL definition.

Example 6–1 C Representation of an STDL Data Type Definition

```
struct data-type-identifier {

c_mapping_for_field_type field-1-name ;

c_mapping_for_field_type field-2-name ;

.
                    .
                    .
} ;
```

A field definition that refers to a previously defined data type identifier has the following format:

struct *data-type-identifier field-name*.

All comments in the data type definition appear in the generated C header file.

The STDL compiler represents DEPENDING ON as an array definition with the number of repetitions being defined by the application program:

```
TYPE rec_id IS RECORD
      number_id INTEGER ;
      array_id ARRAY SIZE n TO m DEPENDING ON number_id OF type ;
END ;
```

The values n and m represent numbers. Simple arrays within structures (records) are not null terminated.

6.2.4 STDL Identifier to C Identifier Conversion

The C identifiers are derived from the corresponding STDL identifiers according to the following rules:

- All uppercase Latin characters are converted to lowercase.

- Hyphens (-) are converted to underscores (_).

- No other characters are changed.

The C language does not support Kanji characters in identifiers.

6.3 Specifying the Call Attributes String

The C client passes to the C input adapter call attributes as a pointer to a string using a procedure generated by the STDL compiler. The function declaration for specifying the call attributes string is:

void *group*_set (char * *attributes_string*)

The value for *group* is the converted name of the task group. The *attributes_string* argument represents values passed to the output adapter.

The values that you can pass to the output adapter vary with each output adapter (see Section 2.3). The information passed on the call is kept by the input adapter on a per-thread/per-group basis. If a thread calls tasks in more than one group, the thread must call the *group*_set procedure for each group for which it intends to set the call attributes. For example, if a thread calls ACMS applications, any login information set for one application is not used when calling another application.

If the client program provides a new call attributes string, the previous call attributes string is overwritten. If the client program provides a null string for the call attributes string, then subsequent calls made to that group from that thread will have no call attributes string.

If an error is encountered in the *group*_set procedure, the error is returned in the `einfo` structure (see Section 6.5).

6.3.1 Calling ACMS Tasks with Call Attributes

To call ACMS tasks (that is, use the ACMS Gateway adapter `acmsda`) with call attributes, your source code creates a string with the call attribute information. For example:

```
char call_attr[100] = "application:add_acms_appl,
                       node:nodea,
                       authentication:usera:mypassword";
```

────────────────── **Note** ──────────────────

Do not use spaces in the string.

──

Use the *group*_set function generated when you compiled the STDL task group specification to supply the address of the call attribute string. For example:

```
add_acms_appl_set(&call_attr);
```

When the client program calls a task, the call attribute values specified in the `call_attr` array are passed to the output adapter.

6.4 Using the Asynchronous C Interface

The asynchronous input adapter allows a C client to execute asynchronous calls to ACMS tasks . The client thread must allocate the storage for the task arguments either on the heap or in static storage rather than in stack storage. The call made by the client thread returns before the task call completes and can do other work.

6.4.1 Asynchronous Call Thread Use

The output adapter for the asynchronous call executes on a separate thread from the client. After the called task executes, the runtime system executes a **completion routine** specified by the client thread on the same thread used for the output adapter.

6.4.2 Using C Clients with the Asynchronous C Adapter

The client program must include the header file generated by the STDL compiler.

The client thread performs each call to a task as follows:

- Uses the same name that an STDL C client would use.

- Passes two additional arguments as the first and second arguments:

  - The address of the completion routine

    Defines the completion routine as a `__stdcall` procedure that returns a `void` and accepts one argument: a pointer to `void`.

    If the client passes STDL_C_SYNCHRONOUS as the completion routine address, the call is made synchronously and the completion routine is not called.

    If the client passes NULL as the completion routine address, the call executes asynchronously, the output adapter executes on a separate thread, but no callback is made.

  - A pointer to `void` to pass to the completion routine

    This argument is passed to the completion routine, allowing it to establish the required context.

The client thread typically does the following:

- Allocates all of the task arguments on the heap.

- Passes, as the completion routine parameter, the address of a structure containing all of the arguments.

- Checks the `einfo` structure after starting the asynchronous call (see Section 6.5).

  If the client thread receives `einfo` status that indicates no error, the client thread successfully started the asynchronous call, and the runtime system always calls the completion routine.

- In the completion routine, checks the `einfo` structure. (This variable is thread-safe.)

Any error encountered in the execution of the called task is returned to the completion routine in the `einfo` structure.

6.5 Using the einfo Structure to Check Status

A C client executing either synchronous or asynchronous calls uses the `einfo` structure to check STDL EINFO status returned from the task call (see Section 2.4).

6.5.1 Exception Information Header File

The C structure definition that corresponds to the STDL EINFO data type definition is shown in Example 6–2.

Example 6–2 C Structure Definition for the einfo Variable

```
#ifndef EINFO_H
#define EINFO_H
   .
   .
   .
#define einfo tps_rpc_get_einfo()->info

typedef struct
  {
  struct
    {
    int    eclass;
    int    ecode;
```

Example 6–2 (Cont.) C Structure Definition for the einfo Variable

```
    char    eproc[32];
    char    epgroup[32];
    int     esource;
    uuid_t ecgroup;
    } info;
  } tps_rpc_einfo_t;

/* function prototypes */

tps_rpc_einfo_t *tps_rpc_get_einfo();

#endif
```

The members in the `einfo` structure in C correspond to the fields in the STDL
EINFO data type definition. The STDL compiler generates the C-language
version of the STDL EINFO data type for use in C clients and servers.

6.5.2 Exception Class Header File

The `eclass.h` header file contains for each exception class identifier a C `define`
statement using the following format:

#define *class-identifier exception-class-value*

For example:

```
#define  ap_invocation_fault -3
```

The value *class-identifier* is the exception class and *exception-class-value* is
the equivalent integer. Section 2.4 provides a summary of STDL exception
information.

6.5.3 Examining Exception Information in C

C client programs examine the `einfo` structure upon completion of a procedure
call. The runtime system returns an exception by setting the STDL-defined
external variable `einfo`. To determine whether an exception is raised,
the C client program examines the exception data supplied in the `einfo`
external structure when the call returns. For example, use the structure
member `einfo.eclass` to determine whether an exception is returned and use
`einfo.ecode` to process the exception (see Section 2.4.2).

HP TP Desktop Connector software provides a function to retrieve the message
text associated with the value found in the `einfo.ecode` field. For example:

```
printf("Error on task call: %s\n\n", stdl_srtl_translate_ecode(einfo.ecode));
```

6.6 Next Steps

After you test and debug the client, set up the management environment.

1. Use the management GUI to establish runtime connections and parameters.

   - If the client calls ACMS tasks, set up parameters (see Section 9.2.3).

2. Set up error logging for the client (see Appendix A).

3. If the client calls an ACMS task, set up the ACMS Gateway adapter environment (see Section 9.3).

7

Writing Automation Servers and Clients

Client programs developed with an Automation language (for example, Visual Basic and Visual J++) can call:

- Tasks in TDL task groups executing under ACMS software

For Java clients developed using Visual J++, the STDL compiler generates COM objects that you register and import into the Visual J++ project as Java classes. You use in your client application the generated Java classes to set and obtain record attributes and invoke tasks. For information on how to program using Java classes generated from COM objects, refer to the documentation provided with Microsoft Visual J++.

To write an Automation client, follow these general steps:

1. Build an Automation server for the Automation clients to use (see Section 7.1).

2. Code the client calls to tasks through the Automation server following the guidelines in Sections 7.2 and 7.3.

7.1 Steps for Building an Automation Server

Use the STDL compiler to generate an adapter stub containing an Automation input adapter and an output adapter that can call your TP application. Using this stub, create an **in-process** Automation server. An automation server converts Automation client calls to the format of call designated by the output adapter in the stub.

To build an Automation server, perform the following steps:

1. Copy to the client development system the task group specification and any related files from an ACMS system (see Section 2.1).

2. Compile the STDL task group specification.

_____ **Note** _____

For ease of use, you can perform the compile and link steps using the
client build utility GUI (see Chapter 5).

_____

On the STDL compile command line, specify input and output adapters
with the -a flag. See Section 4.2 for the complete stdl command syntax.
Use auto for the input adapter and supply the appropriate output adapter
from those shown in Table 7–1.

Table 7–1 Automation Server Output Adapters

Specification	Purpose
acmsda	Calls ACMS tasks using a gateway

For example:

```
stdl -a auto:acmsda add_acms_appl
```

The command creates an Automation input adapter and an ACMS Gateway
output adapter in the generated adapter stub. The compilation produces
files with names in the following format:

group_auto_out_adapter.obj

group_auto.res

The format conventions are:

group	Converted name of the compiled STDL task group specification. When developing client interfaces for ACMS applications, the *group* name prefix is the ACMS *application_name* prefix (see Section 3.2.3).
out_adapter	Name of one of the output adapters from Table 7–1.
res	Resource file for the Automation server.

See the section titled "Compilation Output" in Section 4.2.

3. Write the Automation server, using STDL-generated Automation code for
   the task group being accessed.

   • Access data structures using STDL-generated methods (see
     Section 7.2).

   • Call tasks using STDL-generated methods (see Section 7.3).

- Handle exceptions from task calls using TPware-supplied objects (see Section 7.2.1).

4. Specifying the /DLL qualifier to the linker to produce the Automation server DLL, link the following items:

   - Objects and libraries that depend on the type of task that the Automation server calls and the output adapter used, as shown in Table 7–2.

Table 7–2   Automation Server Adapter-Dependent Link Input

If Client Calls...	Input File Name	Comment
ACMS tasks (uses the `acmsda` adapter)	*group*_auto_acmsda.obj	The adapter stub for the ACMS application (the STDL task group)
	stdl_acmsda.lib	ACMS Gateway adapter runtime link library

   - The resource file for the task group, with a name in the following format:

     *group*_auto.res

     The value *group* is the converted name of the compiled STDL task group specification.

   - The following libraries:

     ```
     stdl_rtm.lib
     stdl_auto.lib
     ole32.lib
     oleaut32.lib
     uuid.lib
     ```

5. Export the following procedures (these are implemented by the Automation server stub):

   ```
   DllGetClassObject
   DllCanUnloadNow
   DllRegisterServer
   DllUnregisterServer
   ```

6. After an Automation server is built, register it with COM using `regsvr32` to allow it to be used by a client program. For example:

   regsvr32 add_acms_appl_auto_acmsda.dll

This `regsvr32` command on the Automation DLL registers the type library. The name of the type library is in the following format:

*group*_Type_Library

The value *group* is the converted name of the task group specification. For example, registering the `add_acms_appl` Automation server assigns the type library name as follows:

```
add_acms_appl_Type_Library
```

If you rebuild an Automation server and change the STDL task group specification or records used, you must first uninstall the original server with `regsvr32` and the `/u` flag.

7.2 Automation Objects and STDL Support

The HP TP Desktop Connector software provides Automation objects and the STDL compiler generates Automation objects for the client interface. Sections 7.2.1 to 7.2.4 describe the objects and their properties.

7.2.1 Supplied Objects

The following standard Automation objects are supplied with HP TP Desktop Connector software:

```
STDL.stdl_einfo
STDL.stdl_uuid
```

These objects are installed when you install the HP TP Desktop Connector software.

The `STDL.stdl_einfo` object is used to store STDL status information in a Microsoft Automation environment. This object has the following Automation properties described using Visual Basic data types:

```
eclass       AS LONG
ecode        AS LONG
ecode_string AS STRING
eproc        AS STRING
epgroup      AS STRING
esource      AS STRING
ecgroup      AS STDL.stdl_uuid
```

The `STDL.stdl_uuid` object is used to store the STDL universal unique identifier (UUID). This object has the following Automation properties described using a Visual Basic data type:

```
uuid_string AS STRING
```

7.2.2 STDL-Generated Automation Objects

The STDL compiler generates Automation objects based on the contents of the task group specification. The Automation client maps a task group to an Automation object as follows:

- One Automation object for each task group

- A method for each task

- A property named `einfo` that points to an `STDL.stdl_einfo` object that contains exception information for an STDL call (see Section 7.2.1)

- A property named `ecode_string` that contains the text of the message associated with the value of the error code returned in the `STDL.stdl_einfo.ecode` property (see Section 7.5.2).

- A property named `call_attributes` that Automation clients can access to specify call attribute values to pass to communications adapters on each call (see Section 7.4)

- Automation objects for STDL records

The Automation object generated for a group has a name in the following format:

group.Group

The value for *group* is the converted name of the task group (see Section 7.2.3).

Within the group Automation object, the STDL compiler creates a **method** for each task in the task group specification.

HP TP Desktop Connector software uses Automation objects to model STDL records. Because Automation does not directly support records, the STDL compiler generates an Automation object for each record used as an argument by a task in the task group being called.

The Automation object for each record has a name in the following format:

group.record-name

The compiler creates one of these objects in each of the following cases:

- The STDL data type is used as an argument to a task in the task group.

  The value *record-name* is derived from the STDL data type identifier that specifies the record layout.

- The STDL data type identifier is used as a data type for a field within another record object being generated for the task group (X/Open syntax only).

The value *record-name* is derived from the STDL data type identifier that specifies the record layout.

- An embedded record is used as a data type for a field within another record object being generated for the task group.

  The compiler gives the object a name in the following format:

  *group.field-name*_record

  The value *field-name* is derived from the STDL field name.

The compiler creates an object for each argument to be passed to a task. If an STDL record is used for more than one argument or is included in more than one other STDL record, the compiler generates only one STDL record object.

For each field within the record, the object has one Automation property. The property name is derived from the STDL field name and the type of the property depends on the STDL field type (see Section 7.2.4).

You can use an object browser (for example, the Visual Basic object viewer) to examine objects defined by the generated Automation server.

7.2.3 STDL Identifier to Automation Name Conversion

The compiler derives Automation names from the corresponding STDL identifiers according to the following rules:

- All uppercase Latin characters are converted to lowercase.
- Hyphen characters (-) are converted to underscore characters (_).
- No other characters are changed.

7.2.4 Automation Data Type Support

When the STDL compiler generates an Automation object for each STDL record (see Section 7.2.2), each field from an STDL record is represented by a property within the generated record object. The type of the property is determined by the STDL data type of the field as shown in Table 7–3.

Table 7–3 Automation Data Type Mapping

	Automation Data Type	
STDL Data Type	**C**	**Visual Basic**
ARRAY	[1]	[1]
DATE	DATE	DATE
DECIMAL STRING	BSTR	STRING
FLOAT SIZE 4	float	SINGLE
FLOAT SIZE 8	double	DOUBLE
INTEGER SIZE 1	short	INTEGER
INTEGER SIZE 2	short	INTEGER
INTEGER SIZE 4	long	LONG
OCTET	unsigned char	BYTE
RECORD	Record object[2]	Record object[2]
TEXT CHARACTER SET ISO-LATIN-1	BSTR	STRING
TEXT CHARACTER SET ISO-LATIN-2	[3]	[3]
TEXT CHARACTER SET ISO-UCS-2	[3]	[3]
UNSIGNED INTEGER SIZE 1	unsigned char	BYTE
UNSIGNED INTEGER SIZE 2	short	INTEGER
UNSIGNED INTEGER SIZE 4	long	LONG
UUID	BSTR	STRING

[1]Array references are to the base data type starting with an index of zero.

[2]An Automation object for the record is created (see Section 7.2.2).

[3]This data type support is not implemented.

7.3 Calling Tasks from Automation Clients

To call tasks within a task group, write the Automation client to perform the following actions:

1. If the client is written in Visual Basic, import the task group type library having a name in the following format:

    group Type Library for ActiveX Access

2. Create an object for each record to be passed as an argument to the task.

This is done by referencing the record object name (*group.record-name*) in one of the following ways:

- In a `CreateObject` service

  You can use the `Server.CreateObject` method to create objects with session, page, or application scope. Objects with page scope are created each time the user accesses the page and are destroyed when the server finishes processing the current page. After session objects are created, they remain in existence until the current session either is abandoned or times out.

- By using object references in Visual Basic

  For example, for the `add_number` record in the `add_task_group`, the object name is `add_task_group.add_number`.

  If the client calls an ACMS task, the first two arguments are the selection string and status.

3. Fill in data in the record objects as necessary.

   The fields within the record objects are referenced as Automation properties of the created record object (see Section 7.2.2).

4. Create an object for the task group in one of the following ways:

   - By referring to the group object in a `CreateObject` service with a name in the following format:

     group.Group

     The value *group* is the converted name of the task group.

   - By using object references in Visual Basic

     For example, the group object name for `add_task_group` is:

     `add_task_group.Group`

5. Optionally, set the call attributes string (see Section 7.4).

6. Within the task group object, call the task as a method.

7. Check the exception information contained in the group object for the status of the call to the task (see Section 7.5.1).

7.4 Specifying the Call Attributes String

Automation clients specify call attributes by referencing an Automation property on the group object (see Section 2.3). This property has the name `call_attributes`.

When a new group object is created, the `call_attributes` property is null. The Automation client can set the `call_attributes` property for use on subsequent method calls using that group object. If the Automation client provides a new call attributes string, the new string overwrites the old string. If the Automation client sets the call attributes string to null, then subsequent method calls on that object will have no call attributes string.

The Automation input adapter does not interpret the contents of the call attributes string. Any error in the contents of the string is not returned until the next call on that group object. If an error occurs, a runtime error is signaled (see Section 7.5.1).

7.4.1 Calling ACMS Tasks with Call Attributes

To call ACMS tasks (that is, use the ACMS Gateway adapter `acmsda`) with call attributes, your source code sets the property `call_attributes` of the task group object with the call attribute information. For example, if the program creates an instance of a task object named `acms` as follows:

```
Set acms = CreateObject("add_acms_appl.group")
```

The program then simply sets the `call_attributes` property using the method call on the appropriate task group object. For example:

```
acms.call_attributes = ("application:add_acms_appl,node:a,authentication:usra:psswrd")
```

──────────────── **Note** ────────────────

Do not use spaces in the string.

──

When the client program calls a task, the call attribute values specified in the `acms.call_attributes` method call are passed to the output adapter at runtime.

7.5 Automation Errors and Status Checking

Two types of error can be returned when calling a task.

- An Automation runtime error (see Section 7.5.1).

- A task call exception (see Section 7.5.2)

7.5.1 Automation Runtime Errors

Automation runtime error values are returned using a 32-bit number known as a result handle (HRESULT). Microsoft defines the structure of the HRESULT value.

Automation errors returned by the adapter begin at the HRESULT value 0x80041001. To determine an STDL error code from the HRESULT format, subtract 0x80041000 from the Automation error code value:

STDL-error-code = <Automation-error-code-value> - 0x80041000

STDL error codes and their corresponding messages are described in a message file (see Section 2.4.2.1).

7.5.2 Examining Exception Information in Automation

On each task call, the code should check the exception information for the status passed back with the call. If an exception occurs, the adapter sets the `eclass` property within the `STDL.stdl_einfo` object to the nonzero value returned from the task call and the `ecode` property within the `STDL.stdl_einfo` object to a nonzero value representing the STDL error code returned.

If no error occurs, the adapter sets the `eclass` property within the `STDL.stdl_einfo` object to zero. Use the `ecode_string` property on the task group object to extract the message text associated with the value of any error returned. For example:

```
Dim acms As Object
Set acms = CreateObject("add_acms_appl.group")
Call acms.add_task(ss, es, input1, input2, answer)
Msgbox ("ecode string = " & CStr(acms.ecode_string))
```

The `CStr` function extracts the text from the property on the `acms` object.

7.6 Next Steps

After you build and debug the client, set up the management environment for the client.

1. Use the management GUI to establish runtime parameters.

    •   If the client calls ACMS tasks, set up parameters (see Section 9.2.3).

2. Set up error logging for the client (see Appendix A).

3. If the client calls ACMS tasks, set up the ACMS Gateway adapter environment (see Section 9.3).

8

Writing Java Clients

A client program developed with the Java language can call tasks in one of the following environments:

- TDL task groups executing under ACMS

8.1 Overview of Java Client Development

Support is provided for the following kinds of programs developed with the Java language:

- Clients developed using Microsoft Visual J++.
- Clients developed using the Sun Microsystems Java Development Kit (JDK). See the HP TP Desktop Connector SPD for supported versions.

For Java clients developed using Visual J++, the interface is provided through an Automation input adapter and COM objects (see Section 7.1).

For Java clients developed using the JDK, the interface is provided through the generated Java input adapter as described in this chapter. The Java client comprises the application code that you write and STDL-generated Java input adapter code.

Use the STDL compiler to generate an adapter stub containing a Java input adapter and an output adapter designated by the type of task calls that the client makes. The adapter stub is used to build a Java client.

The Java input adapter created as part of the generated adapter stub contains the following code.

- Java Native Interface (JNI) code

  Contained within the DLL that also contains the output adapter, the JNI code is the interface between the Java components of the stub and the TPware runtime.

- Java code

  Created as a collection of classes representing the task group specification and its records, this code is the interface between the client application and the JNI code.

Thus, your client does not invoke the JNI code directly. You use the Java classes in the client to call tasks. The Java code in the input adapter calls the JNI code to convert the task calls to the format designated by the output adapter that you specify for the adapter stub.

To write a Java client, follow these general steps:

1. Generate the adapter stub and related code for the Java client (see Section 8.2).

2. Code the client application including calls to tasks, following the appropriate guidelines (see Section 8.3).

3. Build the Java client and related runtime code (see Section 8.2).

8.2 Steps for Building a Java Client

The Java client can call a task on an ACMS system. To build a Java client with an adapter stub, perform the following steps:

1. Set up Java-specific environment variables.

   In addition to using `stdl_set_version` (see Section 4.1) and setting up your C/C++ programming environment, define the environment variables specific to your Java environment.

   (The .bat-style examples assume that your working directory is `C:\work`, and that the JDK was installed in the `C:\jdk1.1.6` directory.)

   - PATH

     Define in the PATH environment variable the JDK bin directory and the directory containing the JNI DLL that is generated by the build. Place the standard JDK directory in the path before any other JDK environments installed on the machine. For example:

     ```
     set path=c:\jdk1.1.6\bin;c:\work;%path%
     ```

   - INCLUDE

     Define in the INCLUDE environment variable the standard JDK and win32-specific directories. For example:

     ```
     set include=c:\jdk1.1.6\include;c:\jdk1.1.6\include\win32;%include%
     ```

- CLASSPATH

  Define in the CLASSPATH environment variable your working directory and the directory for the standard JDK classes. Place these directories before those for any alternative JDK classes. For example:

  ```
  set classpath=.;..;c:\jdk1.1.6\lib\classes.zip;%classpath%
  ```

  Before actually running the adapter code, specify in the CLASSPATH environment variable the path for the JAR file that is generated by the build (see Section 8.8.1).

2. Optionally, define build environment variables for Java tools that are used (see Table 8–1).

Table 8–1   Optional Build Environment Variables for Java Tools

Name	Purpose
STDL_JAVA_JAR_OPTIONS	Overrides default JAR command line options
STDL_JAVA_JAVAC_OPTIONS	Overrides default JAVAC command line options
STDL_JAVA_JDK_HOME	Specifies the location of the JDK installation root

Use these environment variables only to change default operations for the Java tools that the STDL compiler calls.

3. Copy to the client development system a task group specification and any related files from an ACMS system (see Section 2.1).

4. Compile the STDL task group specification.

   On the STDL compile command line, specify input and output adapters with the –a flag. See Section 4.2 for the complete STDL compiler command syntax. Supply the appropriate input adapter and output adapter from those shown in Table 8–2.

Table 8–2 Java Client Adapters

	Specification	Purpose
Input Adapters	java	The Java classes that represent the application objects present their data record fields as public members.
	javabeans	The Java classes that represent the application objects present their data record fields through accessor methods.
Output Adapters	acmsda	Calls ACMS tasks using a gateway.

For example:

```
stdl -a java:acmsda add_acms_appl
```

The command creates a Java input adapter and an ACMS Gateway output adapter in the generated adapter stub. The compilation produces the following files: †

- A Java archive with a name in the following format:†

  group_out_adapter.jar

- A Java adapter stub with a name in the following format:†

  group_java_out_adapter.obj

The format conventions are:

group   Converted name of the compiled STDL task group specification (see Section 8.3.2.1).

out_adapter   Name of one of the output adapters from Table 8–2.

See the section titled "Compilation Output" in Section 4.2.

5. Write the Java client, using STDL-generated Java classes for the task group being accessed (see Section 8.3).

6. Link the components by producing the DLL for the Java client. Specifying the /DLL qualifier to the linker, link the following items:

- The TPware runtime library:

  stdl_rtm.lib

---

† The value*group* is a converted name (see Section 8.3.2.1).

- Objects and libraries that depend on the type of task that the client calls and the output adapter that the client uses, as shown in Table 8–3.

Table 8–3 Java Client Adapter-Dependent Link Input

If Client Calls...	Input File Name	Comment
ACMS tasks (uses `acmsda` adapter)	*group*_java_acmsda.obj	The adapter stub for the ACMS application (the STDL task group)
	stdl_acmsda.lib	ACMS Gateway adapter runtime link library

After you perform the link operations and before you run the client, see Section 8.8 for other steps to perform.

8.3 Java Classes and STDL Support

The HP TP Desktop Connector software provides the native runtime environment. The STDL compiler generates Java classes and the Java Native Interface (JNI) for the client. Sections 8.3.1 to 8.3.3 describe the objects and their properties. See Sections 8.3.3 through 8.6 for information on coding the Java client.

8.3.1 Einfo Java Class and Access Support

The Java class `Einfo` is supplied for each task group to store STDL status information in a Java environment. The task group class has a property called `einfo` with a property type of `Einfo`.

The support for this class depends on whether you use `java` or `javabeans` as the input adapter in the `-a` flag on the STDL command line.

- For the `java` input adapter, this class has the following public fields:

```
public int    eclass
public int    ecode
public String ecodeString
public String eproc
public String epgroup
public int    esource
public String ecgroup
```

- For the `javabeans` input adapter, this class has the following public accessor methods:

```
int getEclass()              void setEclass(int)
int getEcode()               void setEcode(int)
String getEcodeString()
String getEproc()            void setEproc(String)
String getEpgroup()          void setEpgroup(String)
int getEsource()             void setEsource(int)
String getEcgroup()          void setEcgroup(String)
```

See Examples 8–1 and 8–2 for sample code and Section 8.6 for coding guidelines.

8.3.2 STDL-Generated Java Classes and Methods

The STDL compiler generates Java classes and methods based on the contents of the task group specification. The Java client maps a task group to Java classes and methods as follows:

- Name conversion (see Section 8.3.2.1)

- One Java class for each task group (called a task group class) and an invocation method for each task (see Section 8.3.2.2).

- Classes for STDL records (data type identifiers) and for STDL arguments (workspaces) used by a task in the task group; and support for accessing the record classes created (see Section 8.3.2.3).

- Support for Java clients to specify call attribute values to pass to the output adapter on each call (see Section 8.5).

8.3.2.1 STDL Compiler Name Conversion

When processing the STDL task group specification and related code, the STDL compiler derives the *group* name and class names from STDL identifiers as follows:

- Changes the task group name (*group*) or an STDL record name to lowercase. If you are developing a client interface for an ACMS application, the *group* name prefix is the ACMS *application_name* prefix (see Section 3.2.3).

- Replaces any hyphen characters (-) with underscore characters (_).

- Changes the first character of the name to uppercase.

8.3.2.2 Task Group Class and Methods

Within the Java class generated for a task group, the STDL compiler creates an **invocation method** for each task in the task group specification. To enable a client program to invoke each task in the task group, the task group class *group* contains an invocation method for each task, in the following format.

group.task_name([*argument* [,...]]);

The name *task_name* for each invocation method is the same as the corresponding converted task name. This format is for both the `java` and `javabeans` adapters.

If the client is calling an ACMS application, methods have two extra arguments in the following format:

group.task_name(*string,status*[,[*argument* [,...]]]);

The extra arguments are:

string   A selection string, consisting of one STDL record containing one 256-character field of data type ISO-LATIN-1 text.

status   An extended status string, consisting of one STDL record containing one 80-character field of data type ISO-LATIN-1 text. This extended status is message text associated with an ACMS error returned from the ACMS application.

The task group class encapsulates an `Einfo` class to process exception information returned from a task call (see Section 8.3.1).

8.3.2.3 Record Classes and Methods

HP TP Desktop Connector software uses Java classes to model STDL records. The STDL compiler generates a Java class for each record in a task and a class for each argument to be passed to a task. The Java class for an STDL record has a name in the following format:

record-name_field-name

The value *record-name* is derived from the STDL data type identifier that specifies the record layout according to the conversion rules (see Section 8.3.2.1). The value *field-name* is derived from the STDL field name according to the conversion rules (see Section 8.3.2.1).

The compiler creates one of these record classes in each of the following cases:

- The STDL data type identifier is used as an argument to a task in the task group.

- The STDL data type identifier is used as a data type for a field within another record object being generated for the task group (X/Open syntax only).

  The value *record-name* is derived from the STDL data type identifier that specifies the field record layout according to the conversion rules (see Section 8.3.2.1).

- An embedded record is used as a data type for a field within another record class being generated for the task group.

If an STDL data type identifier is used for more than one argument or is included in more than one other STDL record, the compiler generates only one STDL record class.

Support for fields depends on whether you use `java` or `javabeans` as the input adapter in the `-a` flag on the STDL command line.

- For the `java` input adapter, public fields are created, with names in the following format:

  public *data-type field-name*

  The public field name is derived from the STDL field name according to the conversion rules (see Section 8.3.2.1). The data type *data-type* depends on the STDL field type (see Section 8.3.3).

  To support call attributes, the task group object has a public field defined (see Section 8.5).

- For the `javabeans` input adapter, for each field within the record classes generated, both a `get` and a `set` accessor method are created, with names in the following format:

  set*field-name*(*data-type*)

  data-type get*field-name*( )

  The method name is derived from the STDL field name according to the conversion rules (see Section 8.3.2.1). The data type *data-type* depends on the STDL field type (see Section 8.3.3).

  To support call attributes, the task group object has a property that can be accessed by set and get accessor methods (see Section 8.5).

You can use a class browser in an integrated development environment (IDE) to examine objects defined by the generated Java classes.

8.3.3 Java Data Type Support

When the STDL compiler generates Java classes for STDL records (see Section 8.3.2), each field from an STDL record is given a data type as shown in Table 8–4.

Table 8–4 Java Data Type Mapping

STDL Data Type	Java Data Type	Comment
ARRAY		[1]
DATE	Date	
DECIMAL STRING	String	
FLOAT SIZE 4	float	32-bit IEEE 754
FLOAT SIZE 8	double	64-bit IEEE 754
INTEGER SIZE 1	byte	
INTEGER SIZE 2	short	
INTEGER SIZE 4	int	
OCTET	byte	A signed quantity
RECORD	Record object	[2]
TEXT CHARACTER SET ISO-LATIN-1	String	Unicode characters
TEXT CHARACTER SET ISO-LATIN-2	String	[3]
TEXT CHARACTER SET ISO-UCS-2	String	[3]
UNSIGNED INTEGER SIZE 1	byte	Signed value
UNSIGNED INTEGER SIZE 2	short	Signed value
UNSIGNED INTEGER SIZE 4	int	Signed value

[1]Array references are to the base data type starting with an index of zero.

[2]A Java object for the record is created (see Section 7.2.2).

[3]This data type support is not implemented.

8.4 Calling Tasks from Java Clients

To call tasks within a task group, write the Java client to perform the actions shown in the following examples. Example 8–1 shows a client generated from using the `java` input adapter.

Example 8–1 The Java Adapter Sample add_task Call

```
class AddMtsClient
{
  int add(int x, int y) throws Exception
  {
1   Add_task_group_mts taskGroup = new Add_task_group_mts();
2   Add_number inp1 = new Add_number();
    Add_number inp2 = new Add_number();
    Add_number result = new Add_number();

3   inp1.Data = x;
    inp2.Data = y;
4   taskGroup.add_task(inp1, inp2, result);

5   if (taskGroup.einfo.eclass != 0 || taskGroup.einfo.ecode != 0)
    {
        System.out.println("Einfo error on task call");
        System.out.println("eclass = " + taskGroup.einfo.eclass);
        System.out.println("ecode = 0x" + Integer.toHexString(taskGroup.einfo.ecode));
        System.out.println("eproc = " + taskGroup.einfo.eproc);
        System.out.println("epgroup = " + taskGroup.einfo.epgroup);
        System.out.println("esource = " + taskGroup.einfo.esource);
        throw new Exception("Application error");
    }
6   return result.Data;
  }

  static public void main(String args[]) {
      add_mts_client addMtsClient = new add_mts_client();
      if (args.length != 2) {
          System.out.println("Usage: x y");
      }
      else {
          try {
              int x = Integer.parseInt(args[0]);
              int y = Integer.parseInt(args[1]);
              System.out.println("Answer = " + addMtsClient.add(x, y));
          }
           catch (NumberFormatException e) {
               System.out.println("Invalid Input");
          }
           catch (Exception e) {
               System.out.println(e);
          }
      }
    }
}
```

The items in the following list correspond to the callouts in Example 8–1.

1 Create the object that represents the task group itself. Use the `new` method and refer to the task group class name `Add_task_group_mts`.

2 Create an object for each record to be used by the `Add_task_group_mts`
   object. Use the `new` method and refer to the class name. In this example,
   three `Add_number` objects are created.

3 Fill in the data in the record objects as necessary. Directly access the
   public fields within the record argument.

4 Call the task. Use the `add_task` invocation method within the
   `Add_task_group_mts` task group object, and pass the three `Add_number`
   argument parameters.

5 Directly access the `Einfo` object and its fields in the `Add_task_group_mts`
   task group object to determine the result of the task call.

6 Obtain the numeric result of the task call by directly accessing the output
   `Add_number` object.

Example 8–2 shows a client generated from using the `javabeans` input adapter.

Example 8–2 The JavaBeans Adapter Sample add_number Call

```
1 import java.beans.*;
  import java.io.*;
  class AddMtsClient {
    int add(int x, int y) throws Exception {
        try {
2  Add_task_group_mts addTaskGroupMts =
          (Add_task_group_mts) Beans.instantiate(null, "Add_task_group_mts");
3  Einfo einfo              = (Einfo) Beans.instantiate(null, "Einfo");
   Add_number inp1 =
       (Add_number) Beans.instantiate(null, "Add_number");
   Add_number inp2 =
       (Add_number) Beans.instantiate(null, "Add_number");
   Add_number result =
       (Add_number) Beans.instantiate(null, "Add_number");

4  inp1.setData(x);
   inp2.setData(y);
5  addTaskGroupMts.add_number(inp1, inp2, result);
```

(continued on next page)

Example 8–2 (Cont.) The JavaBeans Adapter Sample add_number Call

```
6   einfo = addTaskGroupMts.getEinfo();
7   if (einfo.getEclass() != 0 || einfo.getEcode() != 0) {
        System.out.println("Einfo error on task call");
        System.out.println("Einfo.eclass = " + einfo.getEclass());
        System.out.println("Einfo.ecode = 0x" + Integer.toHexString(einfo.getEcode()));
        System.out.println("Einfo.eproc = " + einfo.getEproc());
        System.out.println("Einfo.epgroup = " + einfo.getEpgroup());
        System.out.println("Einfo.esource = " + einfo.getEsource());
        throw new Exception("Application error");
    }
8   return result.getData();
   }
   catch (UnsatisfiedLinkError e)
   {
        System.out.println("Unable to locate matching DLL add_task_group_mts_java_mts[_g].dll");
        throw e;
   }
   catch (ClassNotFoundException e) {
        System.out.println("Unable to locate an application class");
        throw e;
   }
   catch (IOException e) {
        System.out.println("Unable to load an application class");
        throw e;
   }
   }

   static public void main(String args[]) {
        add_mts_client addMtsClient = new add_mts_client();
        if (args.length != 2) {
           System.out.println("Usage: x y");
        }
        else {
           try {
               int x = Integer.parseInt(args[0]);
               int y = Integer.parseInt(args[1]);
               System.out.println("Answer = " + addMtsClient.add(x, y));
           }
           catch (NumberFormatException e) {
               System.out.println("Invalid Input");
           }
           catch (Exception e) {
               System.out.println(e);
           }
        }
   }
}
```

The items in the following list correspond to the callouts in Example 8–2.

1 Import the classes.

2 Create the object that represents the task group itself. Use the `Beans.instantiate` method and refer to the task group class name `Add_task_group_mts`.

3 Create an object for each record to be used by the `Add_task_group_mts` object. This includes the implicitly used `einfo` object and objects that are explicitly passed as arguments to the `Add_task_group_mts.add_number` method. Use the `Beans.instantiate` method and refer to the class name. In this example, three `Add_number` class objects and one `Einfo` class object are created.

4 Fill in the data in the record objects as necessary. The fields within the record argument and `Einfo` objects are accessed through their get and set accessor methods.

5 Call the task. Use the `add_number` invocation method within the `Add_task_group_mts` task group object, and pass the three `Add_number` argument parameters.

6 Obtain the contents of the `Einfo` object from the `Add_task_group_mts` object through the `getEinfo` accessor method and update the local `einfo` object with the returned information.

7 Examine the `einfo` object using its accessor methods to determine the result of the task call.

8 Obtain the numeric result of the task call by accessing the output `return` object using its `getData` accessor method.

8.5 Specifying the Call Attributes in a Java Client

A Java client can specify call attributes using one of two techniques depending on whether you use `java` or `javabeans` as the input adapter in the `-a` flag on the STDL command line.

- If you use `java` as the input adapter, directly set the `call_attributes` field (see Section 8.5.1).

- If you use `javabeans` as the input adapter, refer to a String property in the task group class named `call_attributes` (see Section 8.5.2).

The values that you specify in the call attributes depend on the output adapter that the client program uses. For more information on call attributes support for each output adapter, see Section 2.3.

8.5.1 Using Call Attributes with the Java Adapter

If the Java client program is using the `java` input adapter and calling an output adapter that supports call attributes, use the `call_attributes` field on the task group object. The field is defined in the following format:

public String call_attributes

The program creates a string to hold the call attributes. For example, if the program calls an ACMS task:

```
String attr = new String("application:add_acms_appl,node:a,authentication:ua:pwrd");
```

_____ **Note** _____

Do not use spaces in the string.

_____

The program must refer to the task group object created using the task group class generated for that group and then set the `call_attributes` field on the task group object under which the task calls are made. For example:

```
add_acms_appl AddAcmsAppl = new add_acms_appl();
    .
    .
    .
AddAcmsAppl.call_attributes = attr;
```

The values in `attr` are assigned to the task group object `call_attributes` field. For details of the runtime processing of call attributes, see Section 8.5.3.

8.5.2 Using Call Attributes with the JavaBeans Adapter

If you use `javabeans` as the input adapter, refer to a String property in the task group class named `call_attributes`. The `call_attributes` property has the following accessor methods:

setCall_attributes(String)

String getCall_attributes( )

If the Java client program is using the `javabeans` input adapter and calling an output adapter that supports call attributes, use the set accessor method `setCall_attributes` on the task group object. The program can create a string to hold the call attributes to be specified. For example, if the program calls an ACMS task:

```
String attr = new String("application:add_acms_appl,node:a,authentication:ua:pwrd")
```

———————————————— **Note** ————————————————

Do not use spaces in the string.

_____

To specify the attribute values, the program must refer to the task group
object created using the task group class and then supply the string to the set
accessor method. For example:

```
add_acms_appl AddAcmsAppl = (add_acms_appl) Beans.instantiate(null,"add_acms_appl");
   .
   .
   .
AddAcmsAppl.setCall_attributes(attr);
```

For details of the runtime processing of call attributes, see Section 8.5.3.

8.5.3 Runtime Processing of Call Attributes

When a new group object is created, the call attributes are null. Using the
group object, the client can set the call attributes value for use on subsequent
method invocations. If the client provides a new call attributes value, the new
value overwrites the old value. If the client sets the call attributes value to
null, subsequent method invocations on that object have no call attributes
value.

The input adapter does not interpret the contents of the call attributes value.
Any error in the value is not returned until the next invocation on that group
object.

For more information on call attributes support for each output adapter, see
Section 2.3.

8.6 Java Runtime Errors

Java-related runtime error values are returned by one of the following
techniques.

• Throwing an exception

• Throwing an error

HP TP Desktop Connector native environment errors or errors returned from
the application server are returned in the `Einfo` class (see Section 8.3.1). STDL
`ecode` error codes that can be returned and their corresponding messages are
described in Section 2.4.2. The STDL eclass codes are specified in the `eclass.h`
file.

8.6.1 Accessing Error Text in the Java Adapter Environment

In the `java` adapter environment, access the `ecodeString` field directly to retrieve message text related to an error returned from a task call. For example:

```
System.out.println("ecodeString = " + taskGroup.einfo.ecodeString);
```

The field is accessed in the `einfo` object contained in the `taskGroup` object.

8.6.2 Accessing Error Text in the JavaBeans Adapter Environment

In the `javabeans` adapter environment, use the `getEcodeString()` public accessor method in the Einfo Java class to retrieve message text related to an error returned from a task call. For example:

```
einfo = taskGroup.getEinfo();
   .
   .
   .
System.out.println("einfo.ecodeString = " + einfo.getEcodeString());
```

This code could display the following in an error situation:

```
einfo.ecodeString = ACMSDI_INVLOGIN, Invalid login attempt
```

On a successful call, this code displays the following:

```
einfo.ecodeString = Normal completion
```

8.7 IDE Interaction with a Java Adapter

If you use `javabeans` as the input adapter with the `-a` flag, the generated classes provide a partial implementation of the JavaBeans standard. The Java input adapter defines the Java classes that the client uses. Discover these classes by using one of the following:

- A Java-aware IDE environment
- The normal JDK BeanBox

The extent of introspection support depends on the capabilities of the development environment that you use. Default discovery is provided by the STDL compiler naming conventions for methods and properties. Simple BeanInfo classes provide introspection support for the classes generated for the following items:

- One for each task group class
- One for each `Einfo` class

- One for each record class

These BeanInfo classes are additional files in the STDL-generated Java archive.

The following are `javabeans` adapter restrictions:

- No graphic representation of the adapter classes is provided.

- No additional development objects for property editors are provided.

- No additional Bean customizers are provided.

- No implementation of Bean serialization is provided.

8.8 Next Steps

After you build the Java client, set up the client (see Section 8.8.1) and the management environment (see Section 8.8.2).

8.8.1 Java Client Setup

To run and successfully access a TP application, a Java client depends on the Java adapter classes and the Java adapter JNI code.

Keep the Java adapter stub DLL and the JAR file containing the Java adapter classes synchronized. Both files must be generated from the same STDL build so that procedures in the DLL match procedures in the JAR file. If the procedures do not match, an `UnsatisfiedLinkError` is thrown.

Ensure that the Java client can locate and use the Java adapter classes. The JAR file containing the Java adapter classes must be available with other class libraries used by the client. Make the file available by adding its path to the `classpath` environment variable. For example:

```
java –classpath add_acms_appl_acmsda.jar;%classpath% add_acms_client 1 1
```

The command runs a console application in the same directory as the JAR file and explicitly names the JAR file and existing `classpath` environment variable. For convenient use, place the command in a .bat file.

If any task group classes in the JAR file are instantiated, the Java Virtual Machine locates and loads the adapter stub DLL and performs initialization operations. If the DLL location is not in the executable path when the Java client runs, an `UnsatisfiedLinkError` is thrown.

To run the Java client under a debugger, rename or copy the adapter stub DLL and append _g to the name of the DLL. For example:

```
C:\java> copy add_acms_appl_java_acmsda.dll add_acms_appl_java_acmsda_g.dll
C:\java>
```

The Java Virtual Machine requires that the _g be appended to the DLL name.

8.8.2 Management Environment Setup

Set up the management environment for the client. Perform the following steps.

1. Use the management GUI to establish runtime parameters.

    • If the client calls ACMS tasks, set up parameters (see Section 9.2.3).

2. Set up error logging for the client (see Appendix A).

3. If the client calls ACMS tasks, set up the ACMS Gateway adapter environment (see Section 9.3).

9

Managing the Client Interface

This chapter describes procedures you use to manage the environment on a client system running the HP TP Desktop Connector product:

- Supplying management information (see Section 9.1)

- Using the management GUI (see Section 9.2)

- Setting up the ACMS Gateway adapter environment if your client calls ACMS tasks (see Section 9.3)

On a client system, you can log errors (see Appendix A).

9.1 Supplying Management Information

In general, the management GUI provides information to the TPware thread controller and various adapters.

9.1.1 Management Information Sources

Whenever an adapter runtime needs information, it uses the following sources (the first listed has the highest precedence):

1. The call attributes (if any) from the client program

   Call attributes can supply only some of the required information.

2. Local settings

   You supply local settings through the local management GUI. Local settings are used if the information is not or cannot be supplied as a call attribute.

3. Remote settings (if `Sharing` is enabled)

   Through the management GUI, you can name a remote node from which management information is retrieved. The TPware runtime caches the remote settings on the local node.

4. Cached information from the remote node (if `Sharing` is enabled)

If an attempt is made to use settings from a remote node and that node cannot be reached, then the last information (if any) retrieved from that node is used.

9.1.2 Management Information by Groups

The management GUI gives you the ability to customize management settings that are specific to your applications and environment. You can enter settings at two levels:

- A group whose name you supply

- The `Default` group

Entering an attribute for a named group allows you to tailor that management attribute for a specific task group. Entering an attribute for the `Default` group sets up a value that is common for all task groups.

9.2 Using the Management GUI

Find the management GUI in the TPware program group. The utility may have any combination of the following tabs:

- `Computer` for selecting the system on which you want to manage HP TP Desktop Connector components (see Section 9.2.1)

- `Sharing` for specifying either of the following:

  - That this computer uses local settings with the option to share its settings with other clients

  - That the computer you are managing uses shared management settings from a remote node

  See Section 9.2.2.

- `ACMS` for setting parameters for the ACMS Gateway adapter on the computer that you are managing (see Section 9.2.3)

The management tabs available depend on the TPware products and options you have installed.

9.2.1 Computer Settings

The `Computer` tab allows you to select either the local node or a remote node for management. This setting defaults to the local node.

When you change the node, the tabs displayed reflect the options installed on that node. If you choose `Share Local Settings`, all the tabs appear.

The name of the computer being managed appears in the title bar.

9.2.2 Using Shared Settings

The `Sharing` tab allows you to set up a particular node from which multiple clients can read their management settings. If many client nodes need the same management settings, you can reduce setup time on each client node by using the shared-settings feature.

The `Sharing` tab allows you to do the following:

- Choose the local node as the source for the management settings (see Section 9.2.2.1).

- Choose a remote node as the source for the management settings that will apply to the node that you are managing (see Section 9.2.2.2).

9.2.2.1 Using Local Settings

If you click `Use Local Settings`, the management settings are read from the local registry. You can also click `Share Local Settings` if you want to share this node's settings with other client nodes. By sharing settings, you can choose that node as the remote source for management information for another computer that you are managing.

9.2.2.2 Using Shared Remote Settings

If you click `Use Remote Settings`, then you need to supply the `Remote Node Name` on which these management settings exist. In this context, *remote node* means another node on which you have configured management GUI settings and have enabled `Share Local Settings`.

Each time a setting is read from a remote node, that setting is written to a local cache. The local cache is used when the remote node is unreachable.

If you change your remote node, or if you switch from `Use Remote Settings` to `Use Local Settings`, the management GUI warns that all cached entries will be deleted and prompts you to confirm this action. If you confirm, then the GUI deletes all cached settings. If you do not confirm, the GUI redisplays the original settings.

If you select `Use Remote Settings`, but have configured local management settings (including the `Default` group settings), the local settings override the remote settings.

9.2.2.3 Restrictions on Registry Access

On Windows platforms, there are various restrictions that affect how a client user account accesses a remote node's registry at runtime to obtain management information.

The following restrictions apply if the remote node is:

- A server in the domain:

  - When the client is logged into the domain

    All users have access, regardless of whether the guest account on the remote node is enabled or not.

  - When the client is not logged into the domain

    If a user has an account on the remote node, the password for the remote account must match the password for the local account. If the guest account is enabled on the remote node, all users have access, and the password restriction does not apply.

- A server not in the domain:

  If a user has an account on the remote node, the password for the remote account must match the password for the local account. However, if the guest account is enabled on the remote node, all users have access.

The following restrictions apply if the remote node is:

- A workstation in the domain:

  - When the client is logged into the domain

    The user must be a member of the `domain admin` group, regardless of whether the guest account on the remote node is enabled.

  - When the client is not logged into the domain

    The user must have an account (with matching password) on the remote node and be in the `domain admin` group for that node.

- A workstation not in the domain:

  The user must have an account (with matching password) on the remote node and be in the `domain admin` group for that node.

9.2.3 ACMS Gateway Adapter Settings

The `ACMS` tab allows you to provide information related to the ACMS Gateway adapter. The STDL task group (for which you provide this information) is generated by the ACMSADU extension and given the ACMS application name as its task group name.

When you click the `ACMS` tab, the `Default` group entry appears in the `Group Name` drop-down box, but the ACMS Gateway adapter settings fields for `Default` remain empty until you enter the relevant information and click `Save`. Configuring the `Default` group allows the runtime system to use these default settings if it cannot find an entry for a particular task group.

The ACMS Gateway adapter settings that you need to provide are:

- `Server Node`: The TCP/IP node on which the ACMS Gateway resides. Running on an ACMS system, the ACMS Gateway connects the ACMS Gateway adapter in the client on the Windows system to the ACMS tasks running on the OpenVMS system.

- `Username`: The name of the user account on the OpenVMS system that clients use for ACMS calls.

- `Password`: The password for the OpenVMS user account that clients use for ACMS calls. (The password is not echoed.)

When the `ACMS` tab settings of a specified group change, all newly started clients see the changed settings. Clients that are running when the changes take effect do not see the changes. Ensure that you restart all clients that call the group so that the modifications take effect.

If the gateway node uses a TCP/IP port that is different from the default, you also need to provide that information through an environment variable (see Section 9.3.2).

If the client specifies call attributes, they override all management settings.

9.2.3.1 Configuring a New ACMS Group

You need to configure a new ACMS group when the group is the `Default` group or any task group that does not use the default settings. If you want a group to use the `Default` group settings, do not perform the following configuration procedure for that group (the runtime system applies default group information to any group without an entry). The procedures differ for changing the settings of an existing ACMS group (see Section 9.2.3.2).

To configure a new ACMS task group, perform the following steps:

1. Click the `ACMS` tab. The `Default` group automatically appears in the `Group Name` box.

2. If you are configuring the `Default` group or an existing group, omit this step. To configure a new group that does not use default settings, do the following.

   a. Click the `New Group` button. The `New Group` dialog box appears.

   b. In the `New Group` dialog box, enter the group name.

   c. Click the `Add` button to add the group to the `Group Name` list box.

3. Enter specific Server Node, Username, and Password information for the group shown in the `Group Name` box.

4. Click the `Save` button.

9.2.3.2 Changing ACMS Gateway Adapter Group Settings

To change the ACMS Gateway adapter settings of a group, perform the following steps:

1. Click the `ACMS` tab.

2. Select a group from the `Group Name` list box to view its settings.

3. Type in the new information for Server Node, Username, and Password for the group shown in the `Group Name` box.

4. After you make changes, save them by clicking the `Save` button.

5. Restart all clients that call the group so the modifications take effect.

If there are any unsaved changes, a dialog box asks whether you want to save all changes.

9.2.3.3 Deleting ACMS Gateway Adapter Group Settings

To delete the settings of a group, perform the following steps:

1. Click the `ACMS` tab.

2. Select the group in the `Group Name` list box.

3. Click the `Delete` button.

4. Restart all clients that call the group so the modifications take effect.

If a group does not appear in the `Group Name` list box, that group uses the settings for the `Default` group, provided that you have entered the `Default` group settings.

9.3 Setting Up the ACMS Gateway Adapter Environment

If your client calls ACMS tasks, you may need to do the following:

- Specify an ACMS Gateway adapter error log file (see Section 9.3.1).
- Specify a TCP/IP port number (see Section 9.3.2).

9.3.1 Specifying an ACMS Gateway Adapter Error Log File

To diagnose problems with communications between the ACMS Gateway adapter and the HP TP Web Connector Gateway for ACMS, specify a file on the client system in which the runtime system writes exception information. In the process in which the client runs, define the following environment variable with the full path and name of a file that holds the exception information:

STDL_ACMSDA_LOG

If the environment variable is defined when the client starts, the runtime system creates the file. If a file of the same name exists, the software appends information to it. Examine the file with any editor that can read ASCII text.

These exceptions are described in the file acmsda_client_messages.txt located in the installation directory.

On a Windows system, the directory has a name in the following format:

install-directory\stdl\include

For example, if you installed the product in the c:\tpware directory, then the client message file is located at:

c:\tpware\stdl\include\acmsda_client_messages.txt

9.3.2 Specifying a TCP/IP Port Number for Calls to ACMS Tasks

If the OpenVMS node running the gateway process defines a TCP/IP port number different from the default , the client process needs to know that port number.

To specify the gateway node's TCP/IP port number in the client process, define the following environment variable with the *node-name* of the node running the gateway process:

STDL_ACMSDA_PORT_*node-name*

For the value of the environment variable, supply the integer number of the port used on the gateway node.

A

TPware Error Logging

TPware software supports recording of runtime errors that cannot be returned to the client process. The software attempts to log these errors and any additional information required to help diagnose the problem indicated by the errors.

A.1 Error Logging Overview

If the TPware software encounters an error at runtime, it attempts to return the error to the client program. If the error cannot be returned to the client, or if additional information should be logged to help diagnose a primary error, TPware attempts to log the primary error and any additional information. The software uses one of the following types of error logging:

- Error logging to a platform event log facility

  By default, TPware software writes error information to a central event log facility specific to the platform (see Section A.2).

- Error logging to a file

  If you define `stdl_log_file` as a system environment variable, TPware software writes error information to the TPware error log file defined by the environment variable (see Section A.3).

The `stdlog` utility is used to view records in a TPware error log file (see Section A.4).

You can use the `STDL_SOURCE_TRACE` environment variable to have information about the source of the error included in the log. If you set the environment variable to the value TRUE, the runtime system includes in the log the number of the source line and the name of the file containing the application source related to the error.

A.2 Use of the Platform Event Log Facility

TPware software writes error information to a platform event log facility if you do not log errors to a TPware error log file. The results depend on the availability of an event log facility:

- If the client process runs on a platform that can log errors to a central event log facility, TPware software writes the error record to the event log facility.

- If the client process runs on a platform that has no event log facility, TPware software writes the error record to a TPware error log file.

  View the records in this file with the `stdlog` utility (see Section A.4).

A.3 Enabling and Disabling Error Logging to a File

Enable TPware error logging to a file by defining a system environment variable named `stdl_log_file`. Supply as a value for `stdl_log_file` the specification of the file to which the error information is to be written.

A.3.1 Enabling Error Logging to a File on Windows Systems

On a Windows 2000 system, if the system environment variable `stdl_log_file` is defined, error information is logged to the TPware file. Define a system environment variable as follows:

1. From the desktop, double click `My Computer`.

2. In the `My Computer` window, double click `Control Panel`.

3. In the `Control Panel` window, double click the `System` icon.

4. In the `System Properties` window, click the `Environment` tab.

5. In the `Environment` window under the `System Variables` list, do the following:

   a. Click on an existing system environment variable.

   b. Click in the `Variable` text box and type `stdl_log_file`.

   c. Click in the `Value` text box and enter the full path and name for the error log file for that system. The value entered for `stdl_log_file` must be a valid path and file name. For example:

   ```
   d:\tpwaresamples\myerror.log
   ```

   d. Click the `Set` button.

   e. Click the `OK` button.

6. Reboot the system for the change to take effect.

TPware errors are subsequently logged to the file specified by the
`stdl_log_file` environment variable. If the file does not exist, the software
creates it. If the file exists, error logging information is appended to it.

A.3.2 Disabling Error Logging to a File

To disable error logging to the TPware error log file, delete the system
environment variable named `stdl_log_file`. If the system environment
variable `stdl_log_file` is not defined, the results depend on the operating
system (see Section A.1).

A.4 Viewing Records in the TPware Error Log File

If error logging to a file is enabled on the client system (see Section A.3), use
the TPware `stdlog` utility to view records in the TPware error log file.

A.4.1 Error Log Utility Syntax

Open a `Command Prompt` window and run the `stdlog` utility.

Synopsis
Use the following syntax:

stdlog [*option-flag* ...]

You can use the `stdlog` utility in interactive mode or single command mode.

If you enter the `stdlog` command alone or without the `-l` flag, the utility
enters interactive mode. If the environment variable `stdl_log_file` is defined,
the software opens and displays the related file. If the environment variable is
not defined, the utility runs and displays the following prompt:

Enter the log file name:

At the file name prompt, enter the name of the file in the current directory and
press `Enter`. If the file is not in the current directory, enter the full path. The
utility displays the specified file and redisplays the prompt.

At the prompt, press `Ctrl + C` to stop the utility and return control to the
`Command Prompt` window.

If you specify the `stdlog` command and option flags, you can issue the entire
command in one line. The utility executes the single command and returns
control to the command prompt.

Option Flag Default Values

All default options are applied to extracting information from the log file. The names and default values of the option flags are shown in Table A–1.

Table A–1   stdlog Option Flags and Default Values

Option Flag	Default
-b [*date-time*]	Displays records with all times before the current date and time of day.
-l *log-file-name*	Displays records in the file specified by the definition of the `stdl_log_file` environment variable. Otherwise, prompts for the name of the log file to access.
-s [*date-time*]	Displays records with all times since the earliest date and time of day.

Option Flags
-b [*yyyy-mm-dd* [*-hh*:*mm* [*:ss* [.*ttt*]]]]

Displays error records that have been logged before the specified absolute date and time of day, expressed in Coordinated Universal Time (UTC) format where:

yyyy	Year from 0000 to 9999
mm	Month of the year from 01 to 12
dd	Day of the month from 01 to 31
hh	Hour of the day from 00 to 23
mm	Minutes from 00 to 59
ss	Seconds from 00 to 59
ttt	Thousandths of seconds from 000 to 999

If you omit the flag, the software displays all records before the current date and time of day.

If you specify –b, but omit the date and time of day, the software displays the records before 00:00:00 on the current date.

If you specify –b and a date, but omit the time, the software displays the records before 00:00:00 on the specified date.

-l *log-file-name*

Displays the specified log file *log-file-name* from the current working directory. If the file is not in the current working directory, specify the full path of the file to display.

If you omit the flag and the `stdl_log_file` environment variable is not defined, the utility prompts for a file name.

-s [*yyyy-mm-dd* **[-***hh***:***mm* **[:***ss* **[.***ttt* **]]]]**

Displays error records that have been logged since the specified absolute date and time of day, expressed in UTC format where:

yyyy	Year from 0000 to 9999
mm	Month of the year from 01 to 12
dd	Day of the month from 01 to 31
hh	Hour of the day from 00 to 23
mm	Minutes from 00 to 59
ss	Seconds from 00 to 59
ttt	Thousandths of seconds from 000 to 999

If you omit the flag, the software displays all error records since the earliest date and time of day.

If you specify -s, but omit the date and time of day, the software displays error records since 00:00:00 on the current date.

If you specify -s and the date, but omit the time of day, the software displays error records since 00:00:00 on the specified date.

A.4.2 Sample Commands and Output

Example A–1 shows a command to select error records from a file located in the current working directory using a time interval specified by -s and -b flags.

Example A–1 Specifying a Time Interval with the `stdlog` **Utility**

```
C:\tpwarelogdir> stdlog -l my.log -s 1998-09-09 -b 1998-09-10
   .
   .
   .
C:\tpwarelogdir>
```

The file `my.log` in the current directory is opened. The command displays error records written during the 24-hour period since 00:00:00 on September 9, 1998 and before 00:00:00 on September 10, 1998.

A sample of the stdlog output is shown in Example A–2.

Example A–2   stdlog Utility Sample Output

```
**********************************************************************
STDL Log Report
d:\tpwaresamples\myerror.log
1998-09-21-09:43:09.810
**********************************************************************
SELECTION CRITERIA:
Before:
Since:
Format: SYSTEM
Type:
**********************************************************************
**********************************************************************
USER: SYSTEM
TIME: 1998-09-20-15:43:09.543
POSTED BY: Client Process
PID: 483
TYPE: Fault
SEVERITY: error
%STDL-E-SRTLCALLGRPBAD, Group init error
**********************************************************************
```

The display consists of heading lines and one record for each TPware error enclosed by lines of asterisk (*) characters. The heading lines contain:

- The full path of the error log file

- The date and time of the day on which the stdlog utility created the report

- Selection criteria used

  - On the Before: line, the date and time of day on a -b flag.

  - On the Since: line, the date and time of day on a -s flag.

The record for each error shows:

- On the USER: line, the name of the user running the client program when the error occurred.

- On the TIME: line, the day, date, time of day, and year when the record was posted.

- On the PID: line, the process identification of the program posting the error record.

- On the TYPE line, the Fault error type. Only faults are logged.

- On the SEVERITY: line, the text error for all errors, primary and secondary.

- On the line under the SEVERITY: line, the text describing the error, with any symbolic error code.

The format for the date and time of day is:

yyyy-mm-dd-hh:mm:ss.ttt

yyyy	Year from 0000 to 9999
mm	Month of the year from 01 to 12
dd	Day of the month from 01 to 31
hh	Hour of the day from 00 to 23
mm	Minutes from 00 to 59
ss	Seconds from 00 to 59
ttt	Thousandths of seconds from 000 to 999

Index