SunXTL 1.1 Remote Client Manager Guide

The Network Is the Computer™

Sun Microsystems Computer Company
2550 Garcia Avenue
Mountain View, CA 94043 USA
415 960-1300 fax 415 969-9131

Part No.: 802-4935-11
Revision A, December 1995

Contents

Figures

Tables

Preface

This book documents the SunXTL Remote Client Manager (RCM).

Note that all references to Solaris in this book apply only to the SPARC version of Solaris.

Who Should Use This Book

This book is intended for Solaris programmers who are knowledgeable in C++ and have an understanding of the SunXTL framework, which is described in the *Sun XTL 1.1 Architecture Guide*. You should be familiar with the SunXTL API to understand the syntax and semantics of the requests and indications. Refer to the *Sun XTL 1.1 Application Programmer's Guide* for more details on the SunXTL API.

How This Book Is Organized

Chapter 1, "Introduction," introduces concepts of the Remote Client Manager and how it works with the SunXTL architecture.

Chapter 2, "RCM Requests and Indications," presents a high-level description of the Remote Client Manager requests and indications.

Appendix A, "Remote Client Sample Sessions," provides sample sessions between a remote client and the Remote Client Manager.

Related Books

The following books are part of the SunXTL documentation set:

- *Sun XTL 1.1 Architecture Guide*
- *Sun XTL 1.1 Administrator's Guide*
- *Sun XTL 1.1 Application Programmer's Guide*
- *Sun XTL 1.1 Provider Programmer's Guide*

What Typographic Changes and Symbols Mean

Table P-1 describes the type changes and symbols used in this book.

Table P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `system% You have mail.`

`AaBbCc123`	What you type, contrasted with on-screen computer output
	
	Also used to highlight class methods in tables
		`system% su`
		`Password::`
		
		`virtual XtelPProvider();`

AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type `rm` *filename*`.`
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in *User's Guide*. These are called *class* options. You *must* be root to do this.

Code samples are included in boxes and may display the following:

`%`	UNIX C shell prompt	`system%`

Table P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
$	UNIX Bourne and Korn shell prompt	`system$`
#	Superuser prompt, all shells	`system#`
`Highlight`	A highlighted table row means the method must be overriden with your implementation code.	`virtual void cleanup()=0;`

Introduction 1

The SunXTL™ 1.1 Remote Client Manager (RCM) provides an Internet Protocol (IP) interface to SunXTL Teleservices. The RCM enables remote clients to use SunXTL Teleservices (excluding data streams) over the network.

In SunXTL 1.0, the Remote Client Manager was named `xtlparser`. The Remote Client Manager provides the full functionality of `xtlparser`, and is fully compatible with `xtlparser`.

RCM and SunXTL

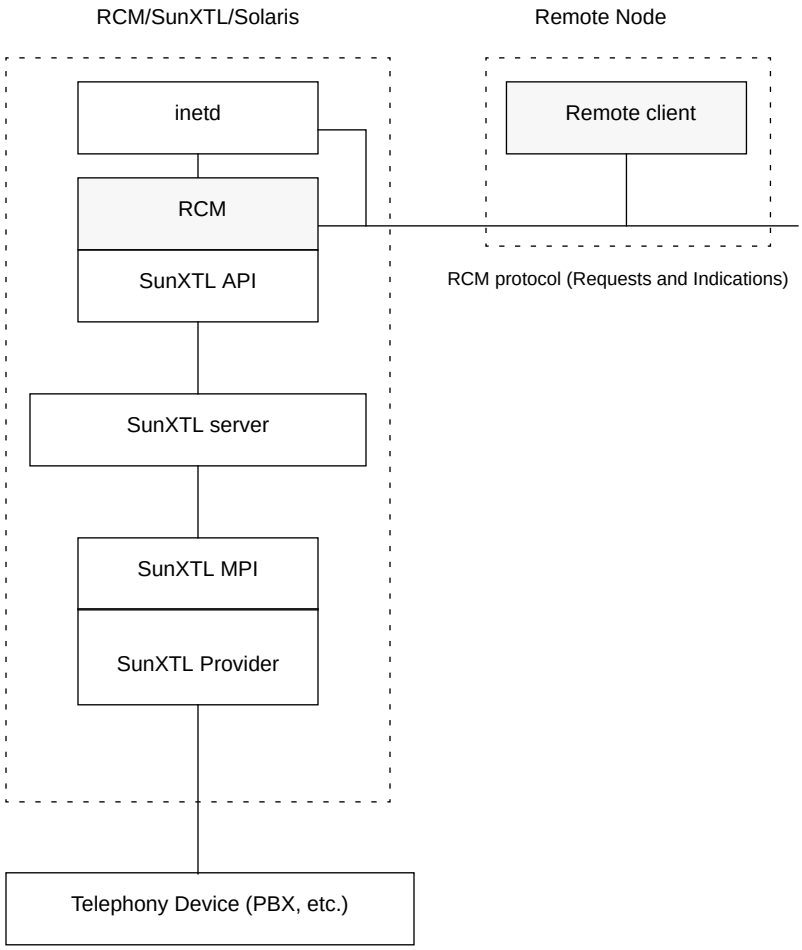
Since the Remote Client Manger (RCM) is a SunXTL application, it should be installed on the SPARC machine where SunXTL Teleservices is installed. Refer to the SunXTL 1.1 CD insert for more details on the RCM and SunXTL 1.1 installation. After installing the RCM and SunXTL 1.1, you should configure the RCM as described in the `README` file located in `/opt/SUNWxtl/src/rcm`.

The RCM runs as a daemon with a port number of 4242. It provides a set of requests and indications for remote clients to communicate with it. The RCM's requests and indications correspond to a subset of the requests and events of the SunXTL API. The RCM's requests and indications are described in Chapter 2, "RCM Requests and Indications". Refer to the *Sun XTL 1.1 Application Programmer's Guide* for more details on SunXTL API.

Remote clients communicate with the RCM, first, by opening a TCP network connection to the RCM, and then sending the RCM's requests over the connection for call control management. The RCM sends indications to the clients in response to the requests it receives.

Appendix A, "Remote Client Sample Sessions," shows sample sessions of a client sending requests to the RCM. The next section describes the software components of the RCM and the SunXTL architecture shown in Figure 1-1.

Figure 1-1 Remote Client Manager and SunXTL Architecture



Software Components

The software components of the Remote Client Manager, remote clients, and SunXTL architecture, shown in Figure 1-1, are

- Remote client
- RCM
- SunXTL API
- SunXTL server
- SunXTL MPI
- SunXTL provider
- Telephony Device.

Remote Client

The Remote Client manages incoming and outgoing calls. It does call control management by sending the RCM's requests over the network. Note that the remote client runs on a remote node. The remote node can be a Solaris/SPARC platform, a Windows/IBM PC platform, or any other platform. The client can establish a connection with the RCM irrespective of the platform type as long as it has a means to open a TCP/IP network connection to the Solaris/SPARC machine on which the RCM is installed.

Chapter 2, "RCM Requests and Indications," provides the details of the RCM's requests and indications.

RCM

The Remote Client Manager interprets the requests sent by the client, and maps the requests to the SunXTL API. It sends indications to the client in response to the requests.

`inetd`

`inetd` is the Internet superserver in Solaris. `inetd` accepts TCP connections from remote clients on the address (port number) of the RCM, and forks RCM processes. There is one instance (process) of RCM for every remote client.

1

SunXTL API

SunXTL API is the Application Programming Interface of SunXTL Teleservices. It provides connection management and control over telephony and voice services through an object-oriented interface. The API is suitable for implementing telephony applications that make, receive, control, and terminate phone calls. Refer to the *Sun XTL 1.1 Application Programmer's Guide* for more details on SunXTL API.

SunXTL MPI

SunXTL MPI is the Media Programming Interface of SunXTL Teleservices. It is an object-oriented interface for developing SunXTL providers that control and manage telephony devices. It handles message passing and dispatching between SunXTL server and SunXTL providers. Refer to the *Sun XTL 1.1 Provider Programmer's Guide* for more details on SunXTL MPI.

SunXTL Server

SunXTL Server handles interprocess message passing between SunXTL API and SunXTL MPI. It also handles asynchronous event notification between SunXTL API and SunXTL MPI. SunXTL Server is completely transparent to SunXTL applications and SunXTL providers. Refer to *Sun XTL 1.1 Architecture Guide* for more details on SunXTL server.

SunXTL Provider

SunXTL Provider manages and controls a telephony device. It uses SunXTL MPI to communicate with SunXTL server. It uses software libraries associated with the telephony device for communicating with the telephony device. Refer to *Sun XTL 1.1 Provider Programmer's Guide* for more details on developing a SunXTL Provider.

Telephony Device

Telephony device can be either hardware (such as a PBX), software, or some combination thereof, that is capable of communicating with a telephone network.

RCM Requests and Indications 2

Remote clients communicate with the SunXTL Remote Client Manager (RCM) with a set of requests and indications for call control management. The requests and indications correspond to a subset of the requests and events of the SunXTL API. The requests and indications are asynchronous, and not all requests have indications.

Remote clients open a TCP connection to the RCM for exchanging the requests and indications. The clients use a transport layer interface such as sockets for establishing a connection with the RCM. Refer to Appendix A, “Remote Client Sample Sessions”, for more details on establishing a connection with the RCM. This chapter explains the syntax and semantics of requests and indications. You should be familiar with the SunXTL API to fully comprehend the syntax and semantics of the requests and indications. Refer to the *Sun XTL 1.1 Application Programmer’s Guide* for more details on the SunXTL API.

Data Types

The RCM requests and indications use the representations of the `XtlByteArray`, `XtlString`, and `XtlKVList` objects; `XtlByteArray`, `XtlString`, and `XtlKVList` are the utility classes of SunXTL API. Refer to the *Sun XTL 1.1 Application Programmer’s Guide* for more details on the utility classes.

Strings

In RCM requests and indications, `XtlByteArray` or `XtlString` are represented as a sequence of characters enclosed in double quotes. A double quote can be embedded in a string if preceded by a backslash. The syntax for a string is as follows:

Strings are noted as "`provider_name`", while empty strings are entered as "".

Numbers

In RCM requests and indications, numbers have `u_long` precision, and are used as integers such as `0` or `31415926`:

Lists

In RCM requests and indications, `XtlKVList` objects are represented as lists. The basic syntax for a list is as follows:

```
((keystring1 value1) (keystring2 value2)...)
```

keystring is a string as described above. *value* is either an integer, string, or another list; an example is given below:

```
(("key1" 1)
 ("key2" "string")
 ("key3" (("key4" 3) ("key5" "last value"))))
```

Enumerations

In RCM requests and indications, enumerations are represented as tokens. For example:

```
Xtl::PROCEEDING_EVENT, an enumeration of the SunXTL API
```

is represented as

```
PROCEEDING_EVENT
```

This symbol should not be in double quotes, nor should it be quoted. The RCM enumerations are equivalent to the SunXTL API enumerations. Refer to the *Sun XTL 1.1 Application Programmer's Guide* for more details on SunXTL API request and event enumerations.

Requests

The RCM has two types of requests:

- Requests that remote clients send to the RCM
- Requests that the RCM sends to remote clients.

Object creation and destruction, provider object, and call objects requests are the requests that the remote clients send to the RCM. Monitoring and filtering requests are the requests that the RCM sends to the remote clients.

2

Object Creation and Destruction Requests

The RCM provides requests for creating and destroying `XtlProvider` and `XtlCall` objects. An `XtlProvider` object represents the service provider that manages connections between a network and a telephone device. An `XtlCall` object represents a call. Refer to the *Sun XTL 1.1 Application Programmer's Guide* for more details on `XtlProvider` and `XtlCall`. The requests for creating and destroying `XtlProvider` and `XtlCall` objects are detailed in Table 2-1.

Table 2-1 RCM Object Create and Destroy Requests

Create and Destroy Object Requests	**Description**
`(create_provider provider_name)`	Creates a `XtlProvider` object for the provider specified by the `provider_name` parameter. The `provider_name` should be a valid provider alias. The `provider_name` should be used in all the requests sent to the provider.
`(destroy_provider provider_name)`	Destroys the `XtlProvider` object of a provider specified by the `provider_name`. The `provider_name` should be a valid provider alias.
`(create_call call_ident provider_name args)`	Creates a `XtlCall` object. The `call_ident` parameter is an arbitrary string chosen by the application to be an identification for the `XtlCall` object. The `provider_name` parameter is the same as the `provider_name` specified with the `create_provider` request. The `args` parameter specifies a `XtlKVList`.
`(destroy_call call_ident)`	Destroys the `XtlCall` object specified by the `call_ident`. The `call_ident` is the arbitrary string chosen by the application to be an identification for a `XtlCall` object.

Provider Object Requests

The RCM provides a set of requests for managing provider objects. Table 2-2 details the RCM requests of a provider object. The RCM provider requests are equivalent to the appropriate `XtlProvider` request methods of SunXTL API. Refer to the *Sun XTL 1.1 Application Programmer's Guide* for the `XtlProvider` request methods of SunXTL API.

Table 2-2 RCM Provider Object Requests

Provider Object Requests	Description
`(list_calls_req provider_name)`	Requests a list of active calls on the provider specified by `provider_name`.
`(enable_offer_event_req provider_name on args)`	Allows the provider specified by `provider_name` to receive call offer events. To register for offer indications, set `on` parameter to `#t`; to unregister set `on` to `#f`. The optional `args` parameter is a list, and can be used to specify provider-specific information.
`(listen_req provider_name listenFor args)`	Allows the provider specified by `provider_name` to listen for a named event specified by `listenFor`. By default provider objects are not registered for any events. The optional `args` parameter specifies provider-specific information.
`(ignore_req provider_name toIgnore args)`	Unregisters an event specified by `toIgnore` with the provider specified by `provider_name`. The optional `args` parameter specifies provider-specific information.
`(extension_req provider_name feature args)`	Requests a provider-specific feature to be activated, such as call forwarding, speed dialing, and so on. `feature` specifies the name of the provider-specific feature, and `args` contains the necessary arguments for the feature.
`(get_call_state_req provider_name call_ref)`	Gets the state of a call specified by `call_ref`. `call_ref` is `XtlCallReference` string that was listed in a `list_calls_ind` indication.

2

Call Object Requests

The RCM provides a set of requests for managing call objects. Table 2-3 details the RCM requests of a call object. The RCM call requests are equivalent to the appropriate `XtlCall` request methods of SunXTL API. Refer to the *Sun XTL 1.1 Application Programmer's Guide* for the `XtlCall` request methods of SunXTL API.

Table 2-3 RCM Call Object requests

Call Object Requests	Description
`(connect_req call_ident local remote media_format args)`	Dials a number to create an outgoing call. The `call_ident` parameter is the string chosen by the application in a `create_call` request. The `local` parameter specifies the address (or telephone number) of the calling party; `remote` is the called party; `media_format` specifies the desired media channel data format. media_format is a list (`XtlKVlist`), and all of the standard constants defined in *<xtl/constants.h>* are valid and have symbol bindings. args is an optional list of provider-specific information.
`(claim_call call_ident call_ref)`	The call specified by `call_ident` takes ownership of an offered call specified by `call_ref`. The `call_ident` parameter is an arbitrary string chosen by the application to be an identification for a `XtlCall` object. The `call_ref` is a string, that corresponds to `XtlCallReference`, extracted from an offer indication.
`(add_to_address_req call_ident addition args)`	Lets you append additional addressing information to the initial address given in a previous `connect_req`. The `call_ident` is the string chosen by the application to be an identification for a `XtlCall` object. The addition parameter specifies a piece of the complete address, and args is an optional list of provider-specific information.
`(answer_req call_ident args)`	Answers, or establishes connection with an incoming call; the `call_ident` parameter is the string chosen by the application in a `claim_call` request. `args` is an optional list of provider-specific information.
`(disconnect_req call_ident args)`	Hangs up a call. The `call_ident` parameter is the string chosen by the application to be an identification for a `XtlCall` object. `args` is an optional list of provider-specific information.
`(hold_req call_ident args)`	Puts a call on hold. The `call_ident` parameter is the string chosen by the application to be an identification for a `XtlCall` object; args is an optional list of provider-specific information.

Table 2-3 RCM Call Object requests *(Continued)*

Call Object Requests	Description
`(unhold_req call_ident args)`	Takes a call off hold. The `call_ident` parameter is the string chosen by the application to be an identification for a `XtlCall` object; `args` is an optional list of provider-specific information.
`(transfer_req call_ident transfer_call_ident args)`	Transfers a call by connecting the call, specified by `call_ident`, to another call specified by `transfer_call_ident`. The `call_ident` and `transfer_call_ident` parameters are the strings chosen by the application in previous requests. The other call must be active; `args` is an optional list of provider-specific information.
`(redirect_req call_ident redirect_to args)`	Redirects the call, specified by `call_ident`, to the address (or telephone number) specified by `redirect_to`. `redirect_to` is a string, and `args` is an optional list of provider-specific information.
`(conference_req call_ident conferee_call_ident args)`	Conferences in another call, specified by `conferee_call_ident`.The `call_ident` and `conferee_call_ident` parameters are the strings chosen by the application in previous requests; `args` is an optional list of provider-specific information.
`(drop_req call_ident args)`	Drops the last call connected to the conference; `args` is an optional list of provider-specific information.
`(offer_req call_ident args)`	Offers the call specified by `call_ident` to other clients and thereby relinquishes ownership of the call if claimed by another client. The `call_ident` parameter is the string chosen by the application to be an identification for a `XtlCall` object; `args` is an optional list of provider-specific information.
`(set_client_state_req call_ident client_state)`	Sets the client state of the call specified by `call_ident` with client-specific information given by `client_state`. The `call_ident` parameter is the string chosen by the application to be an identification for a `XtlCall` object. `client_state` is a list (`XtlKVlist`).

Table 2-3 RCM Call Object requests *(Continued)*

Call Object Requests	Description
`(extension_req call_ident feature args)`	Requests a provider-specific feature; `call_ident` parameter is the string chosen by the application to be an identification for a `XtlCall` object. feature is a string, and `args` is an optional list of provider-specific information.
`(configuration_req call_ident requested_configuration)`	Configures or opens a call's data stream with `requested_configuration`. `call_ident` is the string chosen by the application to be an identification for a `XtlCall` object. `requested_configuration` is a list. `args` is an optional list of provider-specific information.
`(generate_dtmf_req call_ident digits args)`	Generates one or more DTMF tones. `call_ident` parameter is the string chosen by the application to be an identification for a `XtlCall` object. `digits` is a string, and `args` is an optional list of provider-specific information.

Monitoring and Filtering Requests

The RCM has one request for monitoring remote clients and another request for filtering messages. Table 2-4 describes the two requests.

Table 2-4 RCM Monitoring and Filtering Requests

Monitoring and Filtering Requests	Description
`(Parser::heartbeat_req)`	The RCM sends `heartbeat_req` to remote clients periodically to check whether remote clients are alive or not. The remote client should send an indication, `heartbeat_ind`, as a response to the request. The `heartbeat_ind` is described in"Monitoring and Filtering Indications". The RCM determines the frequency of the `heartbeat_req` based on the arguments specified at its start-up time; refer to `/opt/SUNWxtl/src/rcm/README` for more details on the RCM's input arguments.
`(Parser::notify_req)`	The RCM sends a `notify_req` to clients immediately after sending a `Provider::activated_ind`. `Provider::activated_ind` is a provider indication, and is described in "Provider Object Indications". The clients can send a `notify_ind`, a filtering indication, to specify the events to be filtered by the provider object. The `notify_ind` is described in "Monitoring and Filtering Indications".

Indications

The RCM sends indications to remote clients in response to the requests it receives from the remote clients. Also the RCM expects indications from the clients for the requests it sends to the clients. Note that requests and indications are asynchronous, and not all requests have indications.

Provider Object Indications

The RCM sends indications for the provider object requests it receives. Table 2-5 lists the RCM provider object indications. The RCM provider indications are equivalent to the appropriate `XtlProvider` indication methods of the SunXTL API. Refer to the *Sun XTL 1.1 Application Programmer's Guide* for the `XtlProvider` indication methods of the SunXTL API.

Table 2-5 Provider Object Indications

Provider Object Indications	Description
`(Provider::activated_ind provider_name args)`	Confirms that the provider object is created for a `create_provider` request. `provider_name` is the provider alias specified with a `create_provider` request, and `args` is an optional list.
`(Provider::deactivated_ind provider_name args)`	Confirms that the provider object is destroyed for a `destroy_provider` request. `provider_name` is the provider alias specified with a `create_provider` request. `args` is an optional list.
`(Provider::list_calls_ind provider_name call_list)`	Lists active calls of the provider. `provider_name` is the provider specified in a `list_calls_req`. `call_list` is a list of `call_states`. A `call_state` is represented as a list where the key is the name of an `XtlCallState` accessor function, and the value is the appropriately-typed value from the `XtlCallState`.
`(Provider::enable_offer_event_ind provider_name on);`	Confirms the `enable_offer_event_req` if the `on` parameter is returned as `#t`. `provider_name` is the provider alias specified in a `enable_offer_event_req`.
`(Provider::listen_ind provider_name event)`	Confirms that the provider is registered to receive the event specified by event. `provider_name` is the provider alias specified in a `listen_req`. event is `CallEvent` enumeration. Refer to *Sun XTL 1.1 Application Programmer's Guide* for more details on `CallEvent`.
`(Provider::ignore_ind provider_name event)`	Confirms that the provider will no longer receive the event specified by event. `provider_name` is the provider alias specified in a `listen_req`. event is `CallEvent` enumeration. Refer to *Sun XTL 1.1 Application Programmer's Guide* for more details on `CallEvent`.

Table 2-5 Provider Object Indications *(Continued)*

Provider Object Indications	Description
`(Provider::get_call_state_ind provider_name call_state args)`	Returns the call state of a call and any provider-specific arguments. `provider_name` is the provider alias specified in a `get_call_state_req`; `call_state` is represented as a list where the key is the name of an `XtlCallState` accessor function, and the value is the appropriately typed value from the `XtlCallState`. `args` is an optional list.
`(Provider::call_event_ind provider_name call_state event cause args)`	Returns the event that has occurred on the call specified by the `call_state` value. A `call_state` is represented as a list where the key is the name of an `XtlCallState` accessor function, and the value is the appropriately typed value from the `XtlCallState`. `event` and `cause` are `CallEvent` and `Cause` enumerations, respectively. `args` is an optional list of provider-specific information. When a client receives this indication, the `call_state` argument only provides up-to-date `XtlCallReference` and `CallState` information. If a client needs complete state information, it my invoke `get_call_state_req`. Refer to the *Sun XTL 1.1 Application Programmer's Guide* for more details on `CallEvent`, `Cause`, and `CallState`.
`(Provider::offer_ind provider_name call_state args)`	Notifies the provider that a call (whose `XtlCallReference` is specified in `call_state`) is available to be claimed. `call_state` is represented as a list where the key is the name of an `XtlCallState` accessor function, and the value is the appropriately typed value from the `XtlCallState`. Additional provider-specific arguments are given by `args`.
`(Provider::info_ind provider_name args)`	Notifies that provider state has changed for the provider alias given in `provider_name`. Additional provider-specific arguments are given by `args`. If a client needs complete state information, it my invoke `get_call_state_req`.
`(Provider::error_ind provider_name request error args)`	An error has occurred in a given request for the provider alias given in `provider_name`. `request` and `error` are enumerations of provider requests and errors. Refer to *Sun XTL 1.1 Application Programmer's Guide* for more details on error. `args` contains provider-specific details.
`(Provider::extension_ind provider_name feature args)`	Confirms that the provider has received the message sent by `extension_req`. `provider_name` is the provider alias specified in the `extension_req`. feature is a string, and `args` contains provider-specific details

Call Object Indications

The RCM sends indications for the call object requests it receives. Table 2-6 lists the RCM call indications. The RCM call indications are equivalent to the appropriate `XtlCall` indication methods of the SunXTL API. Refer to the *Sun XTL 1.1 Application Programmer's Guide* for the `XtlCall` indication methods of the SunXTL API.

Table 2-6 Call Object Indications

Call Object Indications	Description
`(Call::activated_ind call_ident args)`	Confirms the activation of a call object for a `create_call` request, and the call object can receive events. `call_ident` is the string chosen by the application in the `create_call` request. `args` is an optional list of provider-specific information.
`(Call::deactivated_ind call_ident args)`	Confirms the deactivation of a call object for a `destroy_call` request. The call object can no longer receive events; `call_ident` is the string specified in the `destroy_call` request; `args` is an optional list of provider-specific information.
`(Call::configuraton_ind call_ident configuration)`	Confirms a new configuration for a `configuration_req`. `call_ident` is the string specified in the `configuration_req`. `configuration` is a list that contains the new media channel configuration.
`(Call::event_ind call_ident event cause args)`	Indicates arrival of an event with a related cause code. `call_ident` is the string chosen by the application in a request. `event` and `cause` are `CallEvent` and `Cause` enumerations respectively; `args` is an optional list of provider-specific information. Refer to the *Sun XTL 1.1 Application Programmer's Guide* for more details on CallEvent and Cause.
`(Call::error_ind call_ident request error args)`	Indicates an error condition for a request. Shows the possible errors values. `call_ident` is the string chosen by the application in a request, and `args` is an optional list of provider-specific information. Refer to the *Sun XTL 1.1 Application Programmer's Guide* for more details on `error`.

Table 2-6 Call Object Indications *(Continued)*

Call Object Indications	Description
`(Call::extension_ind call_ident feature args)`	Indicates an `extension indication` for a `extension_req`; it returns the newly activated provider-specific `feature` and its argument list. `call_ident` is the string specified in the `extension_req`, `feature` is a string, and `args` is an optional list of provider-specific information.
`(Call::detect_dtmf_ind call_ident value args)`	Indicates a DTMF event. `call_ident` is the string chosen by the application in a request. `value` is a string, and `args` is an optional list of provider-specific information.
`(Call::configuration call_ident configuration)`	Returns the requested configuration information in response to a `get_configuration` request. `call_ident` is the string specified in the `get_configuration`, and `configuration` is a list.

Monitoring and Filtering Indications

The RCM receives monitoring and filtering indications from remote clients in response to monitoring and filtering requests. Table 2-7 describes monitoring and filtering indications.

Table 2-7 RCM Monitoring and Filtering Requests

Monitoring and Filtering Requests	Description
`(heartbeat_ind)`	Remote clients should send a `heartbeat_ind` in response to a `heartbeat_req` from the RCM. If the RCM does not receive a `heartbeat_ind` within a time-out period, it assumes that the client has exited and the RCM will exit. The time-out period is determined by the arguments given to the RCM at the start-up time; refer to `/opt/SUNWxtl/src/rcm/README` for more details on the RCM's input arguments.
`(notify_ind args)`	If remote clients do not want to receive all the events sent by a provider object by default, the clients should send a `notify_ind`. The clients can send a `notify_ind` at any time after receiving a `notify_req` from the RCM. The RCM sends a `notify_req` to clients immediately after sending a `Provider::activated_ind`. `args` is a list and it can have one of the two following forms: (client_ident ident _number) or (client_ident ident_string) `client_ident` is the string specified by the application. `number` is a number, and `ident_string` is a string. After receiving the indication the RCM filters out client-bound messages which do not contain a key-value pair specified in the `args`.

Errors

The RCM reports errors in two ways: the first mechanism is the normal `error_ind` indication that is sent to the appropriate object as described in provider object and call object indications. The second mechanism is `unknown_object` indication. The RCM sends the `unknown_object` indication when an argument, such as `provider_name` or `call_ident` of a request is not bound to a valid object. The `unknown_object` indication has the following form:

```
(unknown_object request_name object_name)
```

For example,
(`unknown_object` "`create_call`" "`invalid_provider_alias`") indicates that the "`invalid_provider_alias`" argument of the `create_call` request is not a valid provider alias.

Currently, the RCM does not forward the exceptions it receives from the SunXTL API to remote clients. Remote clients should wait for appropriate indications before sending new requests. In particular, after sending a `create_provider` or a `create_call` request, the clients should not send any requests on the new object until the appropriate `activated_ind` is received.

Remote Client Sample Sessions A

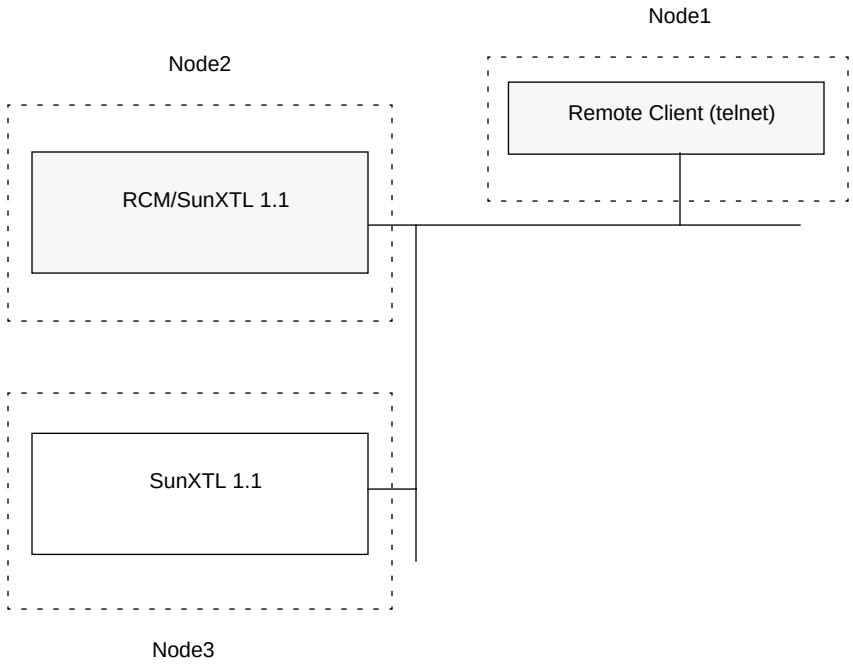
This chapter shows two sample sessions between a remote client and the Remote Client Manager (RCM). The first session shows how the remote client makes an outgoing call, and the second session shows how the remote client receives an incoming call. `telnet(1)` is used as the remote client in the sample sessions.

If you want to develope a remote client, refer to the source code of a remote client example available in the SunXTL 1.1 distribution. After installing the Remote Client Manager package, `SUNWxtlsp`, you can find the source code and a `README` for the remote client example in `/opt/SUNWxtl/src/rcm_client`.

Sample Sessions

`telnet(1)` is used as a remote client in the following sample sessions; both sessions use the Internet Protocol (IP) provider for call management. Refer to the `telnet(1)` and `xtlp_sun_ip(7)` man pages for more details on the telnet and the IP provider. The configuration for the sample sessions is shown in Figure A-1: the remote client (`telnet`) runs on `node1`, the RCM and SunXTL teleservices are installed on `node2`, and SunXTL is installed on `node3`. Note that `node1` does not have SunXTL.

Figure A-1 Configuration for Sample Sessions



Making an Outgoing Call, Local Party Disconnects

In this session the remote client (telnet) manages an outgoing call from `node2` to `node3` using the RCM. Make sure that you run an application on `node3` to receive the call; you can run the `file_play` example program, located in `/opt/SUNWxtl/bin`, with the IP provider on `node3`.

Type the following on node1 to open a connection to the RCM:

```
node1% telnet node2 4242
```

You should see the following if the telnet opens a connection to the RCM successfully:

```
Trying  ...
Connected to node2.
Escape character is '^]'.
```

Now you can send requests to the RCM. Send a request to create an IP provider object:

```
(create_provider "ip")
```

You should not send any requests until you get the following indication from the RCM:

```
(Provider::activated_ind "ip" ())
(Parser::notify_req)
```

Notice that the RCM has sent a `notify_req` immediately after the `activated_ind`. You can send `notify_ind` to filter messages from the provider object. In this session you should not send a `notify_ind` in order to see all the messages from the provider object.

Send a request to create a call object on the IP provider:

```
(create_call "call1" "ip" ())
```

You get the following indication from the RCM:

```
(Call::activated_ind "call1" (("call_reference" "24 xtlp_sun_ip.0")
("state" "IDLE") ("local_address" "") ("remote_address" "")
("media_channel_available" #f) ("owner" #t) ("incoming" #f)
("claimable" #f) ("extended_state" ()) ("client_state" ()) ("format" ())
("display" "")) ())
```

Send a request to establish a call on `node3`; Make sure you are running an application to receive the call on `node3`; you can run `file_play` example program located in `/opt/SUNWxtl/bin` on `node3`.

```
(connect_req "call1" "" "node3" () ())
You get the following event indications from the RCM:
(Call::event_ind "call1" (("call_reference" "24 xtlp_sun_ip.0")
("state" "PROCEEDING") ("local_address" "node2")
("remote_address" "node3") ("media_channel_available" #f) ("owner" #t)
("incoming" #f) ("claimable" #f) ("extended_state" ()) ("client_state" ())
("format" ()) ("display" "")) PROCEEDING_EVENT CAUSE_NORMAL ())
(Call::event_ind "call1" (("call_reference" "24 xtlp_sun_ip.0")
("state" "ALERTING") ("local_address" "node2")
("remote_address" "node3") ("media_channel_available" #f) ("owner" #t)
("incoming" #f) ("claimable" #f) ("extended_state" ()) ("client_state" ())
("format" ()) ("display" "")) ALERTING_EVENT CAUSE_NORMAL ())
(Call::event_ind "call1" (("call_reference" "24 xtlp_sun_ip.0") ("state"
"CONNECTED") ("local_address" "node2") ("remote_address" "node3")
        ("media_channel_available" #f) ("owner" #t) ("incoming" #f)
      ("claimable" #f) ("extended_state" ()) ("client_state" ()) ("format" ())
("display" "")) CONNECT_EVENT CAUSE_NORMAL ())
(Call::event_ind "call1"   (("call_reference" "24 xtlp_sun_ip.0") ("state"
"CONNECTED") ("local_address" "node2") ("remote_address" "node3")
("media_channel_available" #t) ("owner" #t) ("incoming" #f)
("claimable" #f) ("extended_state" ()) ("client_state" ()) ("format" ())
("display" "")) CHANNEL_AVAILABLE_EVENT CAUSE_NORMAL ())
(Call::event_ind "call1" (("call_reference" "24 xtlp_sun_ip.0") ("state"
"CONNECTED") ("local_address" "node2") ("remote_address" "node3")
("media_channel_available" #t) ("owner" #t) ("incoming" #f)
("claimable" #f) ("extended_state" ()) ("client_state" ())
("format" ( ("Encoding" "ULaw") ("SampleSize" 8) ("SampleRate" 8000)))
("display" "")) INFO_EVENT CAUSE_NORMAL ())
```

Configure the call to generate DTMF tones:

```
(configuration_req "call1"  (("INPUT" "STREAM") ("OUTPUT" "STREAM")
("OUTPUT" "DTMF_DETECTOR") ("INPUT" "DTMF_GENERATOR") ))
```

The RCM sends the following indication in response to configuration_req:

```
(Call::configuration_ind "call1" (("call_reference" "24 xtlp_sun_ip.0")
("state" "CONNECTED") ("local_address" "node2") ("remote_address"
"node3") ("media_channel_available" #t) ("owner" #t)
("incoming" #f) ("claimable" #f) ("extended_state" ()) ("client_state" ())
("format" ( ("Encoding" "ULaw") ("SampleSize" 8) ("SampleRate" 8000)))
("display" "")) ( ("INPUT" "DTMF_GENERATOR") ("OUTPUT"
"DTMF_DETECTOR") ("INPUT" "STREAM") ("INPUT_STREAM_FD" 8)
("OUTPUT" "STREAM") ("OUTPUT_STREAM_FD" 8)))
(Call::event_ind "call1" (("call_reference" "24 xtlp_sun_ip.0") ("state"
"CONNECTED") ("local_address" "node2") ("remote_address" "node3")
("media_channel_available" #t) ("owner" #t) ("incoming" #f) ("claimable" #f)
("extended_state" ()) ("client_state" ()) ("format" ( ("Encoding" "ULaw")
("SampleSize" 8) ("SampleRate" 8000))) ("display" "")) INFO_EVENT
CAUSE_NORMAL ())
```

You can now send DTMF tones:

```
(generate_dtmf_req "call1" "123" () )
```

The RCM sends the following indication:

```
(Call::extension_ind "call1" (("call_reference" "24 xtlp_sun_ip.0")
("state" "CONNECTED") ("local_address" "node2") ("remote_address"
"node3") ("media_channel_available" #t) ("owner" #t) ("incoming" #f)
("claimable" #f) ("extended_state" ()) ("client_state" ()) ("format" ( ("Encoding"
"ULaw") ("SampleSize" 8) ("SampleRate" 8000))) ("display" ""))
"DTMF_GENERATOR" ( ("DTMF_STRING" "123")))
```

Similarly you can send other requests for controlling the call. When you no longer need the call you can hang-up the call with a disconnect_req:

```
(disconnect_req "call1"  () )
```

At the end, destroy the call and the provider objects:

```
(destroy_call "call1")
(destroy_provider "ip")
```

Finally close the connection with the RCM:

```
(exit)
```

The RCM closes the connection, and the telnet exits.

```
Connection closed by foreign host
node1%
```

Receiving an Incoming Call, Remote Party Disconnects

In this session the remote client (telnet) manages an incoming call from `node3` to `node2` using the RCM. You have to run an application on `node3` to establish a call from `node3` to `node2`; you can use `dial_pad` example program, located in `/opt/SUNWxtl/bin`, with the IP provider on `node3`.

Type the following on node1 to open a connection to the RCM:

```
node1% telnet node2 4242
```

You should see the following if the telnet opens a connection to the RCM successfully:

```
Trying ...
Connected to node2.
Escape character is '^]'.
```

Now you can send requests to the RCM. Send a request to create an IP provider object:

```
(create_provider "ip")
```

You should not send any requests until you get the following indication from the RCM:

```
(Provider::activated_ind "ip" ())
(Parser::notify_req)
```

To receive incoming calls, enable the provider object to receive call offer events. Send the following request to the IP provider object:

```
(enable_offer_event_req "ip" #t () )
```

You get the following indication from the RCM:

```
(Provider::enable_offer_event_ind "ip" #t)
```

Now if you establish a call from `node3` to `node2` using the IP provider, you should get an offer_ind indication from the RCM. As mentioned earlier, you can use the `dial_pad` example program located in `/opt/SUNWxtl/bin` to establish a call from `node3` to `node2`.

```
(Provider::offer_ind "ip" (("call_reference" "27 xtlp_sun_ip.0")
("state"  "INCOMING") ("local_address" "node2")
("remote_address" "node3") ("media_channel_available" #f)
("owner" #f)
("incoming" #t)  ("claimable" #t) ("extended_state" ())
("client_state" ( ("XtlKVList" "UNKNOWN")))
("format" ( ("XtlKVList" "UNKNOWN")))  ("display" "")) ())
```

After receiving the `offer_ind` indication you should claim the call to take ownership of the call with a `claim_call` request; note that you have to use the call reference specified in the `offer_ind` to claim the call. Here the call reference is "`27 xtlp_sun_ip.0`".

```
(claim_call "call2" "27 xtlp_sun_ip.0")
```

You should get the following `activated_ind` indication for the call object, `call2`, creation:

```
(Call::activated_ind "call2" (("call_reference" "27
xtlp_sun_ip.0")
("state" "INCOMING") ("local_address" "node2")
("remote_address" "node3") ("media_channel_available" #f)
("owner" #t)
("incoming" #t) ("claimable" #f) ("extended_state" ())
("client_state" ())
("format" ()) ("display" "")) ())
```

After claiming the call you should answer the remote party which is trying to establish the call from node3:

```
(answer_req "call2" ())
```

You get the following event indications from the RCM if the call is established successfully:

```
(Call::event_ind "call2" (("call_reference" "27 xtlp_sun_ip.0")
("state" "CONNECTED") ("local_address" "node2")
("remote_address" "node3") ("media_channel_available" #f)
("owner" #t) ("incoming" #t) ("claimable" #f) ("extended_state" ())
("client_state" ()) ("format" ()) ("display" "")) CONNECT_EVENT
CAUSE_NORMAL ())
(Call::event_ind "call2" (("call_reference" "27 xtlp_sun_ip.0")
("state" "CONNECTED") ("local_address" "node2")
("remote_address" "node3") ("media_channel_available" #t) ("owner" #t)
("incoming" #t) ("claimable" #f) ("extended_state" ()) ("client_state" ())
("format" ()) ("display" "")) CHANNEL_AVAILABLE_EVENT
CAUSE_NORMAL ())
(Call::event_ind "call2" (("call_reference" "27 xtlp_sun_ip.0")
("state" "CONNECTED") ("local_address" "node2")
("remote_address" "node3") ("media_channel_available" #t) ("owner" #t)
("incoming" #t) ("claimable" #f) ("extended_state" ()) ("client_state" ())
("format" ( ("Encoding" "ULaw") ("SampleSize" 8) ("SampleRate" 8000)))
("display" "")) INFO_EVENT CAUSE_NORMAL ())
```

Now the call is established between node3 and node2, and you can control the call as you wish by sending other requests and indications to the RCM.

In this session, assume that the remote party from node3 disconnects the call; you get the following indication when the remote party disconnects the call:

```
(Call::event_ind "call2" (("call_reference" "27 xtlp_sun_ip.0")
("state" "DISCONNECTED") ("local_address" "node2")
("remote_address" "node3") ("media_channel_available" #f) ("owner" #t)
("incoming" #t) ("claimable" #f) ("extended_state" ()) ("client_state" ())
("format" ( ("Encoding" "ULaw") ("SampleSize" 8) ("SampleRate" 8000)))
("display" "")) DISCONNECT_EVENT CAUSE_NORMAL ())
(Call::deactivated_ind "call2" (("call_reference" "27 xtlp_sun_ip.0")
("state" "INVALID") ("local_address" "node2") ("remote_address" "node3")
("media_channel_available" #f) ("owner" #t) ("incoming" #t) ("claimable" #f)
("extended_state" ()) ("client_state" ()) ("format" ( ("Encoding" "ULaw")
("SampleSize" 8) ("SampleRate" 8000))) ("display" "")) ())
```

Now you can destroy the call and provider objects:

```
(destroy_call "call2")
(destroy_provider "ip")
```

Finally close the connection with the RCM:

```
(exit)
```

The RCM closes the connection, and the telnet exits:

```
Connection closed by foreign host.
node1%
```

A

Reader Comments

We welcome your comments and suggestions to help improve this manual. Please let us know what you think about the *SunXTL 1.1 Remote Client Manager Guide*, part number *802-4935.-10*

■ The procedures were well documented.

Strongly Agree	Agree	Disagree	Strongly Disagree	Not Applicable
❑	❑	❑	❑	❑

Comments _____

■ The tasks were easy to follow.

Strongly Agree	Agree	Disagree	Strongly Disagree	Not Applicable
❑	❑	❑	❑	❑

Comments _____

■ The illustrations were clear.

Strongly Agree	Agree	Disagree	Strongly Disagree	Not Applicable
❑	❑	❑	❑	❑

Comments _____

■ The information was complete and easy to find.

Strongly Agree	Agree	Disagree	Strongly Disagree	Not Applicable
❑	❑	❑	❑	❑

Comments _____

■ Do you have additional comments about the *SunXTL 1.1 Remote Client Manager Guide*?

_____

_____

_____

_____

Name: _____

Title: _____

Company: _____

Address: _____

_____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 1 MOUNTAIN VIEW, CA

POSTAGE WILL BE PAID BY ADDRESSEE

SUN MICROSYSTEMS, INC.
Attn: Manager, Publications
MS MPK 14-101
2550 Garcia Avenue
Mt. View, CA 94043-9850