



Telestra SIP Service API

Revision A
Version 2
March 2026
Document DOC-TEL-SIP-API-A-2

Advanced Simulation Technology inc.
500A Huntmar Park Drive • Herndon, Virginia 20170 USA
(703) 471-2104 • asti-usa.com

Product Name: Telestra

Telestra SIP Service API

Copyright © 2026 ASTi

Restricted rights: copy and use of this document are subject to terms provided in ASTi's Software License Agreement (www.asti-usa.com/license.html).

ASTi
500A Huntmar Park Drive
Herndon, Virginia 20170 USA

Red Hat Enterprise Linux (RHEL) Subscriptions

ASTi is an official Red Hat Embedded Partner. ASTi-provided products based on RHEL include Red Hat software integrated with ASTi's installation. ASTi includes a Red Hat subscription with every purchase of our Software and Information Assurance (SW/IA) maintenance products. Systems with active maintenance receive Red Hat software updates and support directly from ASTi.

Export Restriction

Countries other than the United States may restrict the import, use, or export of software that contains encryption technology. By installing this software, you agree that you shall be solely responsible for compliance with any such import, use, or export restrictions. For full details on Red Hat export restrictions, go to the following:

www.redhat.com/en/about/export-control-product-matrix

Revision history

Date	Revision	Version	Comments
10/28/2025	A	0	Initial baseline version.
1/14/2026	A	1	Removed proprietary data statement from footer.
3/9/2026	A	2	Formatted all protocol names with the <code>span.keyword</code> style and fixed multiple <code>span.fields</code> and <code>span.data_type</code> styles in tables throughout the document.

Contents

1.0 Introduction	1
1.1 API commands, events, and responses	2
1.1.1 login	3
1.1.2 make-call	3
1.1.3 call-incoming	4
1.1.4 answer-call	5
1.1.5 switch-call	6
1.1.6 get-status	6
1.1.7 hangup-call	7
1.1.8 logout	8
1.2 API errors	9

1.0 Introduction

Session Initiation Protocol (SIP) calling provides comprehensive Voice-over-IP (VoIP) functionality through both component interfaces and programmatic application programming interface (API) control. The Telestra SIP Service API offers a Transmission Control Protocol (TCP)-based API for complete control of calling functions, including call management, account control, and status monitoring. Host applications connect to the Telestra server using the port defined in the SIP plan's [tcp-settings] section (default port 3550).

The API works in conjunction with SIP plans in Studio, referencing SIP accounts defined in SIP plan configuration files. These SIP plans contain account credentials, server settings, and Real-time Transport Protocol (RTP) port ranges that the API uses for all operations. You must configure and assign a SIP plan to your Telestra server before using the API, as detailed in "SIP plans" in the *Studio Technical User Guide*. It also references SIPAccount components, where the API handles call signaling and control, and the components manage audio routing within your communication models. This design allows centralized credential management while supporting dynamic programmatic call control.

To use the SIP Service API, first complete the following steps:

1. Configure your SIP server to accept third-party SIP devices, as defined in "Telestra SIP integration with Cisco® Unified Communications Manager" in the *Studio Technical User Guide*.
2. Add and configure a SIP plan in your Studio project, as described in "Add a SIP plan" and "Edit SIP plan accounts and RTP settings" in the *Studio Technical User Guide*.
3. Assign it to your Telestra server in the **Telestra Editor's SIM SERVER** settings, as described in "Assign a SIP plan to the Telestra server" in the *Studio Technical User Guide*.
4. Add and configure a SIPAccount component to your working model. To learn more about SIPAccount components, go to "SIPAccount" in the *Studio Components Reference Guide*.
5. Verify network connectivity to your SIP infrastructure. Confirm the Telestra server can reach the SIP server's IP address and that SIP ports (typically 5060) and RTP port ranges are accessible through any firewalls.
6. Confirm that host applications can establish TCP connections to the API port.

The API complements Studio component work flows by providing programmatic control of VoIP calls while leveraging your communication models' audio-processing capabilities.

1.1 API commands, events, and responses

All API communication follows a consistent request-response pattern using null-terminated JSON messages. Clients send commands to the Telestra SIP Service API to initiate operations, while the service responds with either responses or events broadcast to all connected clients.

The API supports both synchronous operations that return immediate responses (e.g., `get-status`) and asynchronous event notifications (e.g., `call-incoming` events). This event-driven architecture enables multiple applications to monitor SIP activity simultaneously and implement distributed control systems.

All API communication uses null-terminated JSON strings. Message formatting requires proper JSON structure with null terminators (`\0`) to indicate message boundaries. The following example shows a typical request to the Telestra server:

```
{
  "cmd": "make-call",
  "account": "account-1"
  "num": "3002"
}
```

Commands are specific API operations that clients can execute, such as initiating calls, querying account status, or managing authentication. Telestra currently supports the following commands:

- `login`
- `make-call`
- `answer-call`
- `switch-call`
- `get-status`
- `hangup-call`
- `logout`

The API also broadcasts events to connected clients automatically, such as `call-incoming`, which notifies all clients when an account receives an incoming call. This section defines and provides example commands, responses (if applicable), and asynchronous events.

1.1.1 login

Use `login` to dynamically log into a SIP server using the supplied credentials. This command enables you to log in with credentials that differ from the SIP plan or reconnect a logged-out account.

Field	Type	Description
cmd	string	"login"
account	string	The SIP plan account that's logging in
user	string	The SIP plan account's username
pass	string	The password required to log in
server	string	The SIP server's IP address

Table 1: login request fields

Request

```
{
  "cmd": "login",
  "account": "account-3",
  "user": "3003",
  "pass": "sip3003",
  "server": "10.2.101.23"
}
```

The API does not return a specific response. Failed log-ins generate `fail` events, as described in Section 1.2, "API errors" on page 9 below. Use the `get-status` command and `status` response to check the account's status. Go to Section 1.1.6, "get-status" on page 6 to learn more about this command.

1.1.2 make-call

Use this request to initiate an outbound call from a specified account to a target extension.

Field	Type	Description
cmd	string	"make-call"
account	string	The SIP plan account that's making the call
num	string	The extension to call

Table 2: make-call request fields

Request

```
{
  "cmd": "make-call",
  "account": "account-1",
  "num": "3002"
}
```

In response, the API issues a `call-return` event with an assigned call ID.

Field	Type	Description
event	string	"call-return"
account	string	The SIP plan account that made the call
call_id	integer	The call's ID number

Table 3: make-call response fields

Response

```
{
  "event": "call-return",
  "account": "account-1",
  "call_id": 0
}
```

1.1.3 call-incoming

The API automatically sends a `call-incoming` broadcast event to all connected clients when any account receives an incoming call, enabling multiple applications to monitor SIP activity simultaneously.

Field	Type	Description
event	string	"call-incoming"
account	string	The SIP plan account receiving the call
call_id	integer	The call's ID number
caller	string	The caller's extension
caller_name	string	The caller's name

Table 4: call-incoming event fields

Response

```
{
  "event": "call-incoming",
  "account": "account-2",
  "call_id": 1,
  "caller": "3001",
  "caller_name": "account-1"
}
```

1.1.4 answer-call

Use `answer-call` to accept an incoming call using the call ID provided in the `call-incoming` event.

Field	Type	Description
cmd	string	"answer-call"
account	string	The SIP plan account receiving the call
call_id	integer	The call's ID number

Table 5: answer-call request fields

Request

```
{
  "cmd": "answer-call",
  "account": "account-1",
  "call_id": 0
}
```

In response, the API automatically sends a `call-answered` broadcast event to all connected clients when any account receives an incoming call, enabling multiple applications to monitor SIP activity simultaneously.

Response

```
{
  "event": "call-answered",
  "account": "account-2",
  "call_id": 0
}
```

1.1.5 switch-call

Use `switch-call` to switch between multiple active calls, placing the current call on hold and activating the specified call.

Field	Type	Description
cmd	string	"switch-call"
account	string	The SIP plan account that is switching calls
call_id	integer	The ID of the destination call

Table 6: switch-call request fields

Request

```
{
  "cmd": "switch-call",
  "account": "account-1",
  "call_id": 1
}
```

The API does not respond with a specific event. Subsequent `get-status` queries reflect call status changes.

1.1.6 get-status

Use `get-status` to query the current status of a SIP account, including registration state and active calls.

Field	Type	Description
cmd	string	"get-status"
account	string	The SIP plan account to query

Table 7: get-status request fields

Request

```
{
  "cmd": "get-status",
  "account": "account-1"
}
```

The API returns a `status` event with comprehensive account information:

Field	Type	Description
event	string	"status"
account	string	The SIP plan account in the status report
calls	array	A list of calls this account is on
account_status	integer	A standard SIP status code

Table 8: status response fields

Response

```
{
  "event": "status",
  "account": "account-1",
  "calls": [{"call_id": 5,
             "caller": "3003",
             "callee": "3001",
             "started": true,
             "stopped": false}
            ],
  "account_status": 200
}
```

1.1.7 hangup-call

Use `hangup-call` to terminate an active call for the specified account and call ID.

Field	Type	Description
cmd	string	"hangup-call"
account	string	The SIP plan account that's ending the call
call_id	integer	The call's ID number

Table 9: hangup-call request fields

Request

```
{
  "cmd": "hangup-call",
  "account": "account-1",
  "call_id": 0
}
```

In response, the API automatically sends a `call-ended` broadcast event to all connected clients when any account ends a call, enabling multiple applications to monitor SIP activity simultaneously.

Field	Type	Description
event	string	"call-ended"
account	string	The SIP plan account that ended the call
call_id	integer	The call's ID number

Table 10: call-ended event fields

Response

```
{
  "event": "call-ended",
  "account": "account-2",
  "call_id": 0
}
```

1.1.8 logout

Use `logout` to log out an active SIP account and terminate its registration with the SIP server.

Field	Type	Description
cmd	string	"logout"
account	string	The SIP plan account that's logging out

Table 11: logout request fields

Request

```
{
  "cmd": "logout",
  "account": "account-3"
}
```

The API does not return a specific response event. Subsequent `get-status` queries reflect changes to account statuses.

1.2 API errors

The API reports errors through `fail` events that provide detailed information about command failures and system problems. These events include the affected account and descriptive error messages to help identify and resolve issues:

Field	Type	Description
event	string	"fail"
account	string	The SIP plan account experiencing a failure
error	string	An error message describing the failure

Table 12: API error fields

Response

```
{
  "event": "fail",
  "account": "account-1",
  "error": "Failed to log in to account: account-1"
}
```

Common error conditions include authentication failures from incorrect credentials or unreachable servers, call management errors from invalid call ID numbers or account references, network connectivity issues affecting SIP registration, and command format problems from malformed JSON or missing required fields.

Use `get-status` to monitor for `fail` events and implement appropriate retry mechanisms, connection recovery procedures, and status validation. Go to Section 1.1.6, "get-status" on page 6 to learn more about this command's behavior.