Creatio Academy

Telephony integration (CTI)

Version 7.18

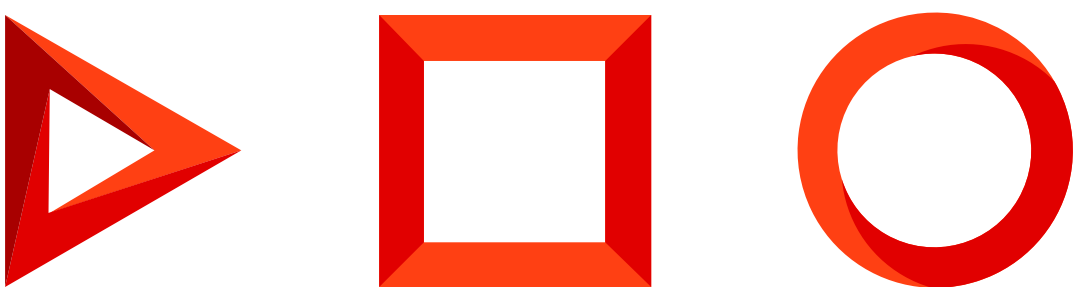


Table of Contents

Telephony integration basics

Advanced

Creatio can be integrated with a number of automatic telephone exchanges (Private Branch Exchange, PBX), which enables users to manage calls directly in Creatio UI. Phone integration functions are available in the form of a CTI (Computer Telephony Integration) panel, as well as the [*Calls*] section. Standard CTI panel functions:

- Displaying incoming calls with contact/account identification by the subscriber's phone number.
- One-click calls initiated from Creatio UI.
- Call management (reply, place on hold, end or transfer call).
- Displaying call history for managing connections of calls to various system records and call follow-up.

All calls made or received are stored in the [*Calls*] section. In this section, you can view when a call was started, when it ended and how long the call was; as well as the list of system records connected to the call.

By default, Creatio cloud has a function for making calls between system users without using any additional software.

Depending on the integrated phone system and specifics of its API (Application Programming Interface), different architectural mechanisms are used. The API also affects available phone integration functions. For example, the call playback function is not available for all phone systems, the web phone is available when integrating with Webitel, etc. Regardless of the phone integration mechanism being used, the CTI panel interface remains the same for all Creatio users.

Phone integration methods in Creatio

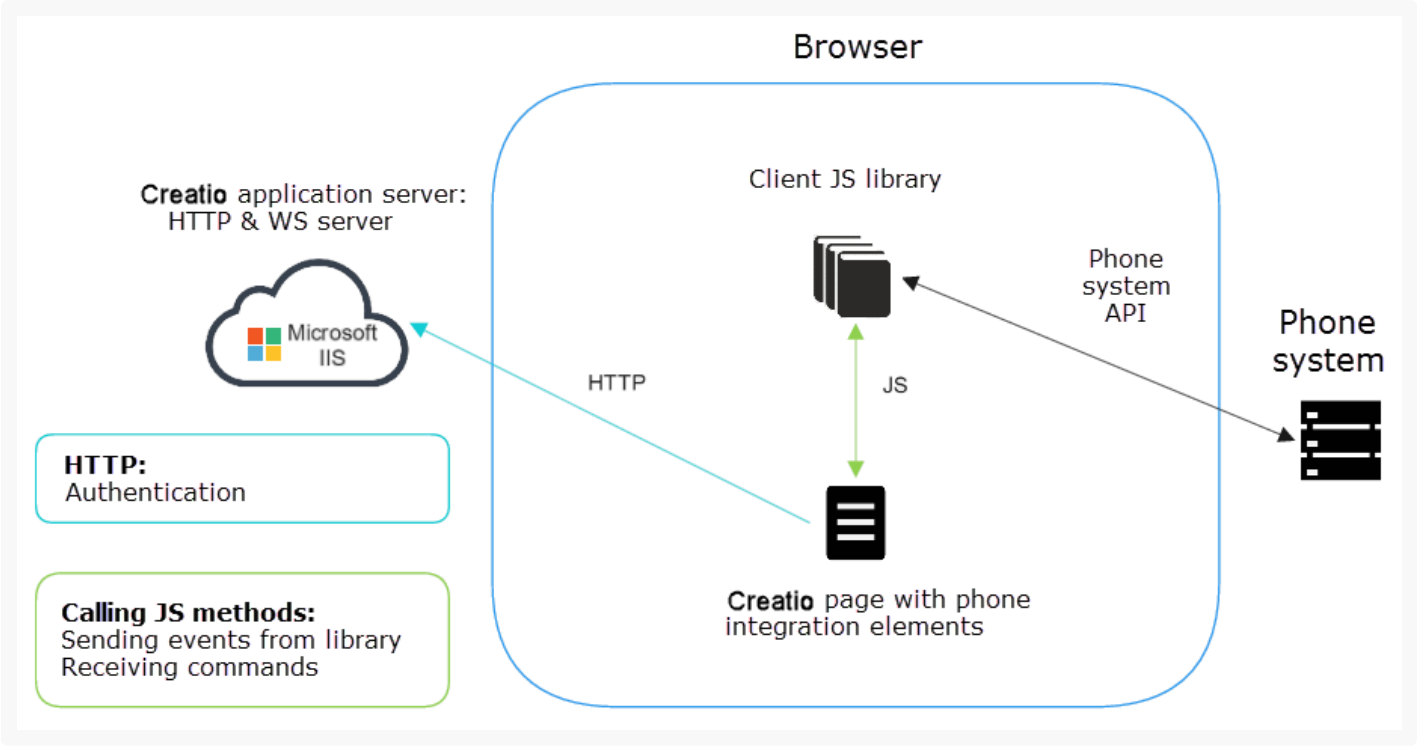
There are two types of integration methods: `first party` and `third party` integrations.

In a `first party` integration each user has a separate integration connection. Phone system events are handled as part of that connection.

For a `third party` integration, a single connection to the prone system server is used for handling phone system events for all users. In a third party integration an intermediate `Messaging Service` link is used for distributing information streams for all users.

JavaScript adapter on the client side

When integrating with JavaScript adapter on the client side, the work with the prone system is done directly from a web browser. Interactions with the phone system and JavaScript-library, usually supplied by the prone system manufacturer, is done through the phone system API. The library broadcasts events and accepts execution commands using JavaScript. In the context of this integration, the Creatio page interacts with the application server for authentication using the HTTP(S) protocol.



This integration method can be used with a first party phone system API, such as Webitel, Oktell, Finesse. Webitel and Oktell connectors use WebSocket as connection protocol, while the Finesse connector uses long-polling http queries.

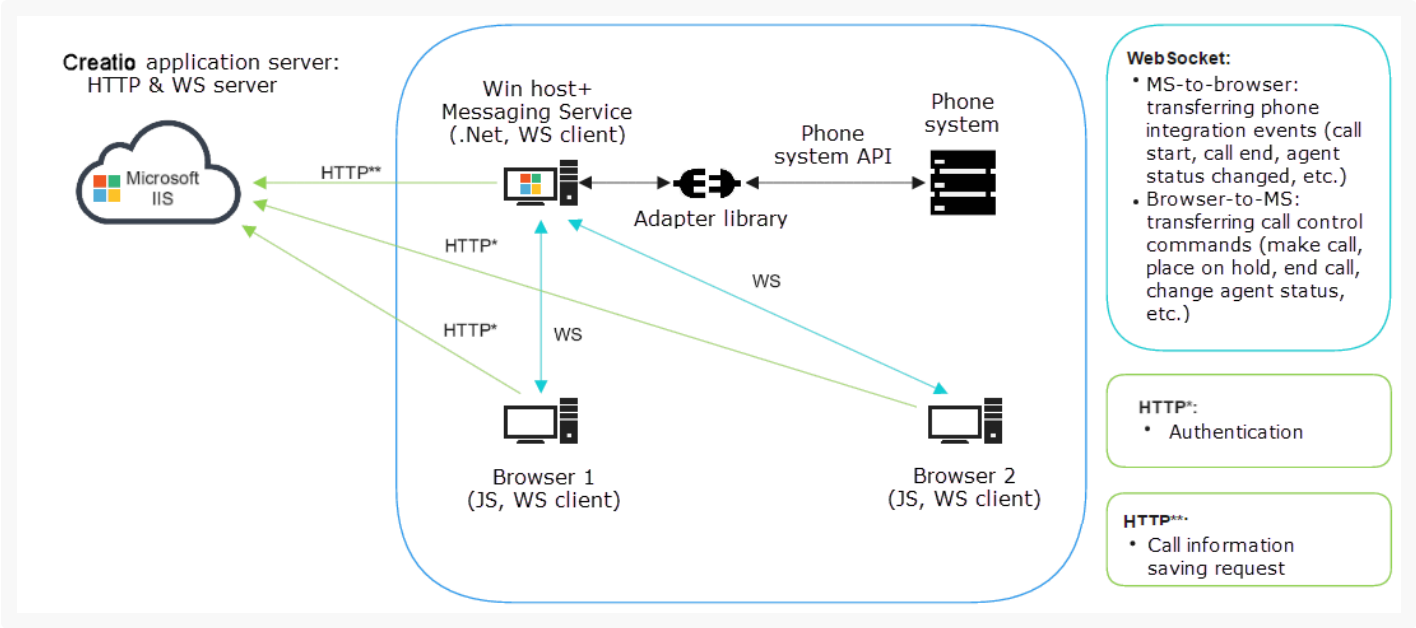
The advantage of the `first party` integration method is that it does not require any additional nodes, such as Messaging Service. Using an integration library, the CTI panel connects directly to the phone system server API from a browser on the user's PC.

For incoming calls the phone server passes the new call start event and call parameters through WebSocket to the client integration library. When receiving a new call command, the library generates the `RingStarted` event that is passed to the application page.

For incoming calls, client part generates the call start command that is passed through WebSocket to the phone integration server.

Terrasoft Messaging Service on the server side

If integrating with Terrasoft Messaging Service (TMS) on the server side, all phone integration events pass through TMS, which interacts with the phone system through the manufacturer's library. The library interacts with the phone system through the API. TMS also interacts with the Creatio application server for executing query for saving call information in the database using HTTP(S). Interaction with a client application, such as passing events and receiving commands, is done via WebSocket. In case of integration with JavaScript adapter on the client side, Creatio page interacts with the application server for authentication, using HTTP(S).



This integration method applies to third party phone system API (TAPI, TSAPI, New Infinity protocol, WebSocket Oktell). This integration type requires `Messaging Service` – a Windows proxy service that works with the phone system adapter library. The `Messaging Service` is a universal phone system library hoster, such as Asterisk, Avaya, Callway, Ctios, Infinity, Infra, Tapi. When receiving client messages, the `Messaging Service` automatically connects used Creatio library and initiates connection to phone system. The Messaging Service is essentially a functional wrapper for those phone integration connectors that do not support client integration for interacting with phone functions in browsers (event generation and handling, data transfer). A user's PC conducts two types of communication:

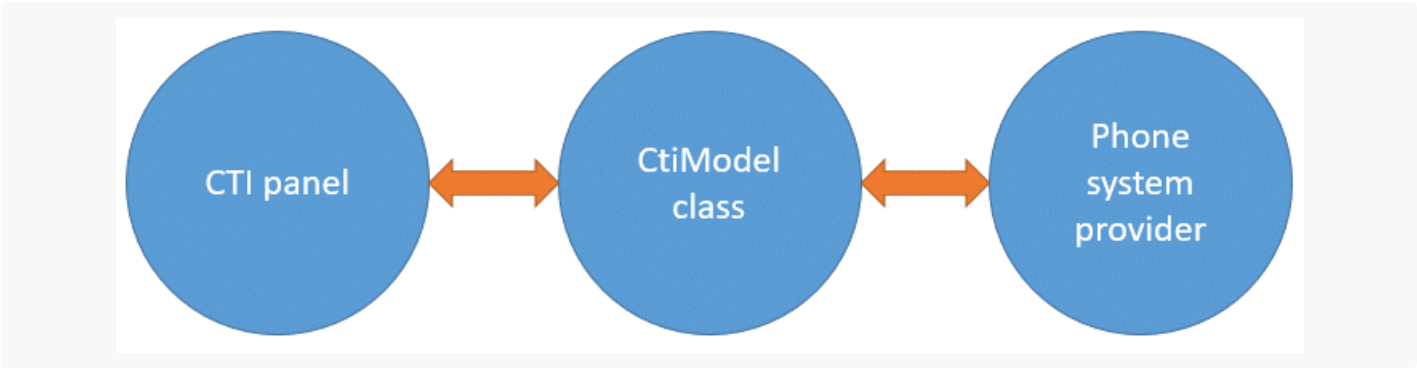
- HTTP connection with Creatio application server for authentication with host on which the *Messaging Service* is installed;
- WebSocket connection for working directly with phone integration.

For incoming calls the phone system passes the new call start event and call parameters through the adapter library. When receiving a new call command, the Messaging Service generates the `RingStarted` event that is passed to the client.

For an outgoing calls, the client generates a call start command, which is passed via WebSocket to the Messaging Service, which generates an outgoing call message for the phone system.

Interaction between the phone connectors and Creatio

All connectors interact with configuration through the CtiModel class. It handles the events received from the connector.



The list of supported class events is provided in table.

Supported events of the CtiModel class

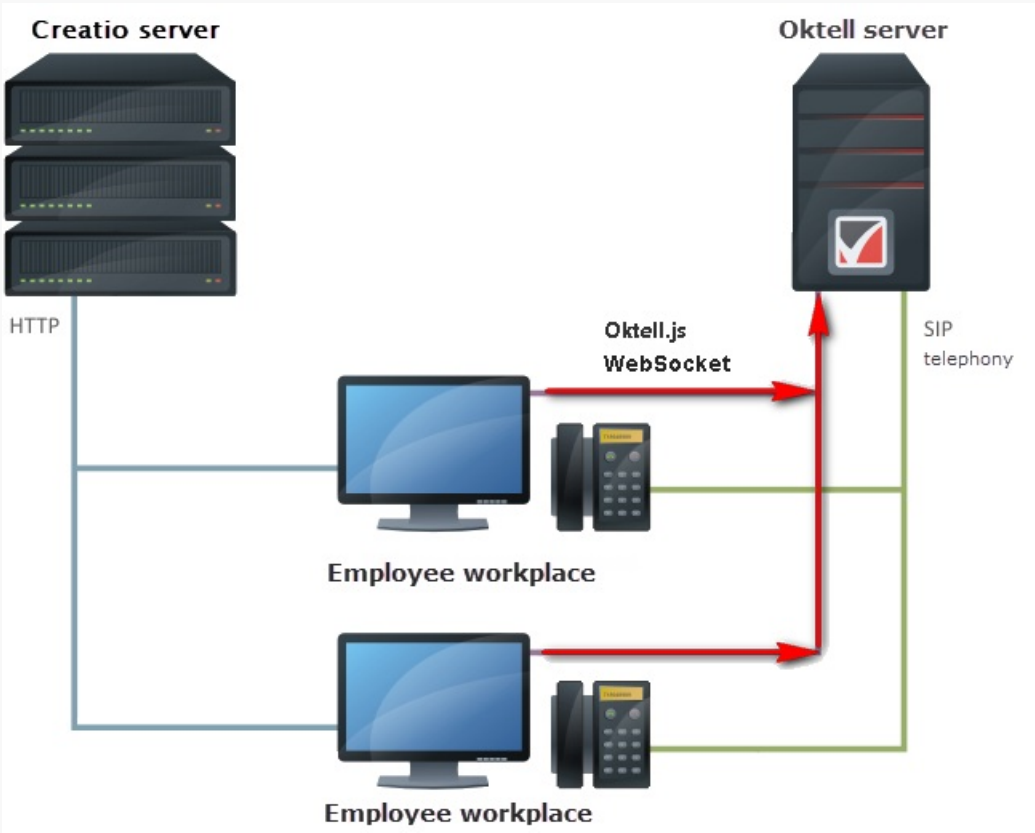
Event	Description
`initialized`	Triggered on completion of provider initialization.
`disconnected`	Triggered on provider disconnection.
`callStarted`	Triggered at the start of a new call.
`callFinished`	Triggered after call completion.
`commutationStarted`	Triggered after establishing call connection.
`callBusy`	Triggered on changing call status to "busy" (TAPI only).
`hold`	Triggered after placing call on hold.
`unhold`	Triggered after resuming a call.
`error`	is triggered on errors.
`lineStateChanged`	Triggered after changing available operations for a line or a call.
`agentStateChanged`	Triggered on changing the agent status.
`activeCallSnapshot`	Triggered on updating the list of active calls.
`callSaved`	Triggered after creating or updating a call in the database.
`rawMessage`	Generic provider event. Triggered on any provider event.
`currentCallChanged`	Triggered on changing the main call. For example, primary call ends during a consultation.
`callCentreStateChanged`	Triggered if an agent enters or exits Call center mode.
`callInfoChanged`	Triggered on modifying a call data by database Id.
`dtmfEntered`	Triggered if Dtmf signals were sent to the phone line.
`webRtcStarted`	Triggered on a webRtc session start.
`webRtcVideoStarted`	Triggered on a webRtc video stream session start.
`webRtcDestroyed`	Triggered on a webRtc session end.

Integration with Oktell

Advanced

Oktell integration with Creatio is implemented on the client level using the oktell.js library. The oktell.js source code is located in the `OktellModule` configuration schema of the `CTIBase` package.

The Oktell server communicates with phones and with the end clients (browsers). With this integration method Creatio does not requires its own WebSocket server. Each client connects via the WebSocket Protocol directly to the Oktell server. The Creatio application server creates pages and provides data from the application database. There is no direct relationship between Creatio and Oktell server. Access is not required, so customers process and combine the data of the two systems independently. The Oktell web client and the oktell.js plugin, embedded in other projects, are implemented according to this principle.



Oktell.js

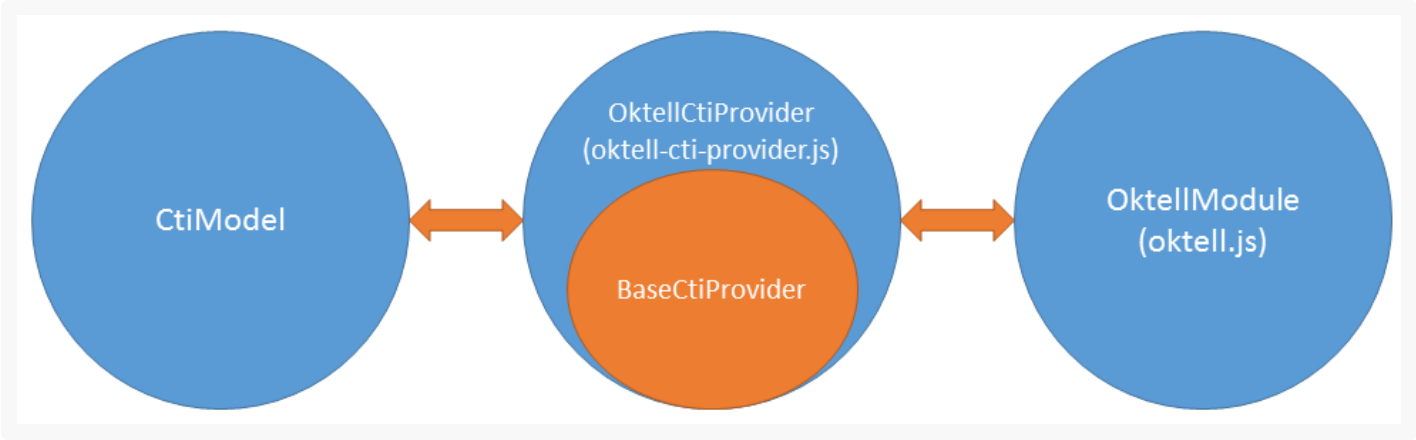
Oktell.js is a javascript library for embedding the functionality of the call control in a CRM system. Oktell.js uses the Oktell WebSocket Protocol to connect to the Oktell server. The advantage of this Protocol is the establishing of a permanent asynchronous connection to the server, which enables you to receive events from the server Oktell and execute certain commands. Because the Oktell WebSocket protocol is quite complicated to implement, the Oktell.js wraps the WebSocket Protocol methods inside itself thus providing simple management functionality.

Voice transmission between subscribers

In a conversation between the oktell and Creatio operators, voice is transmitted via the Session Initiation Protocol (SIP). This requires that either the VoIP phone or the Softphone operator be installed on your computer.

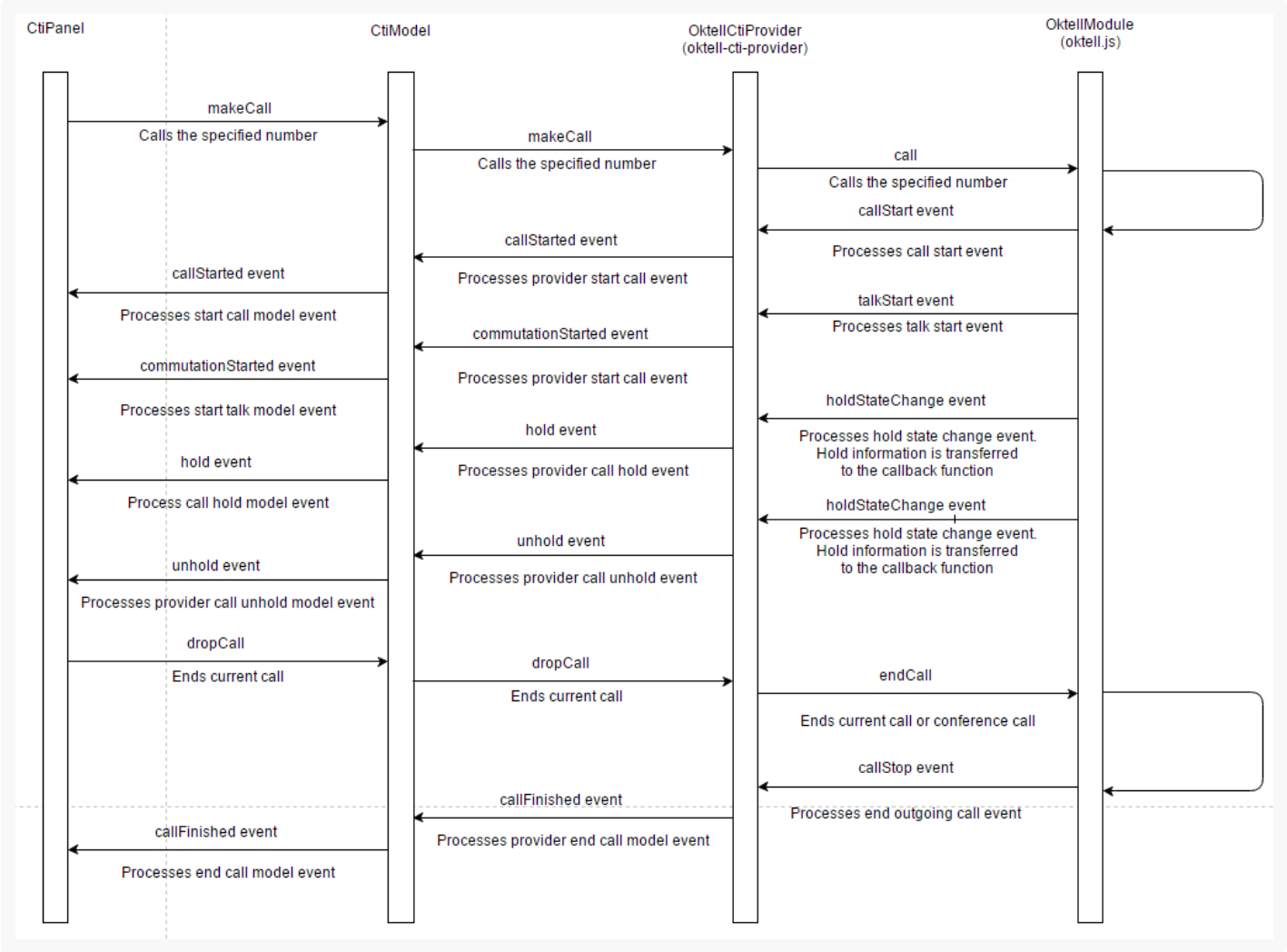
Interaction of components

The interaction with the oktell.js library is executed via the *OktellCtiProvider* class, which is a link between *CtiModel* and *OktellModule* that contains the oktell.js code. The `OktellCtiProvider` class implements the `BaseCtiProvider` interface class.

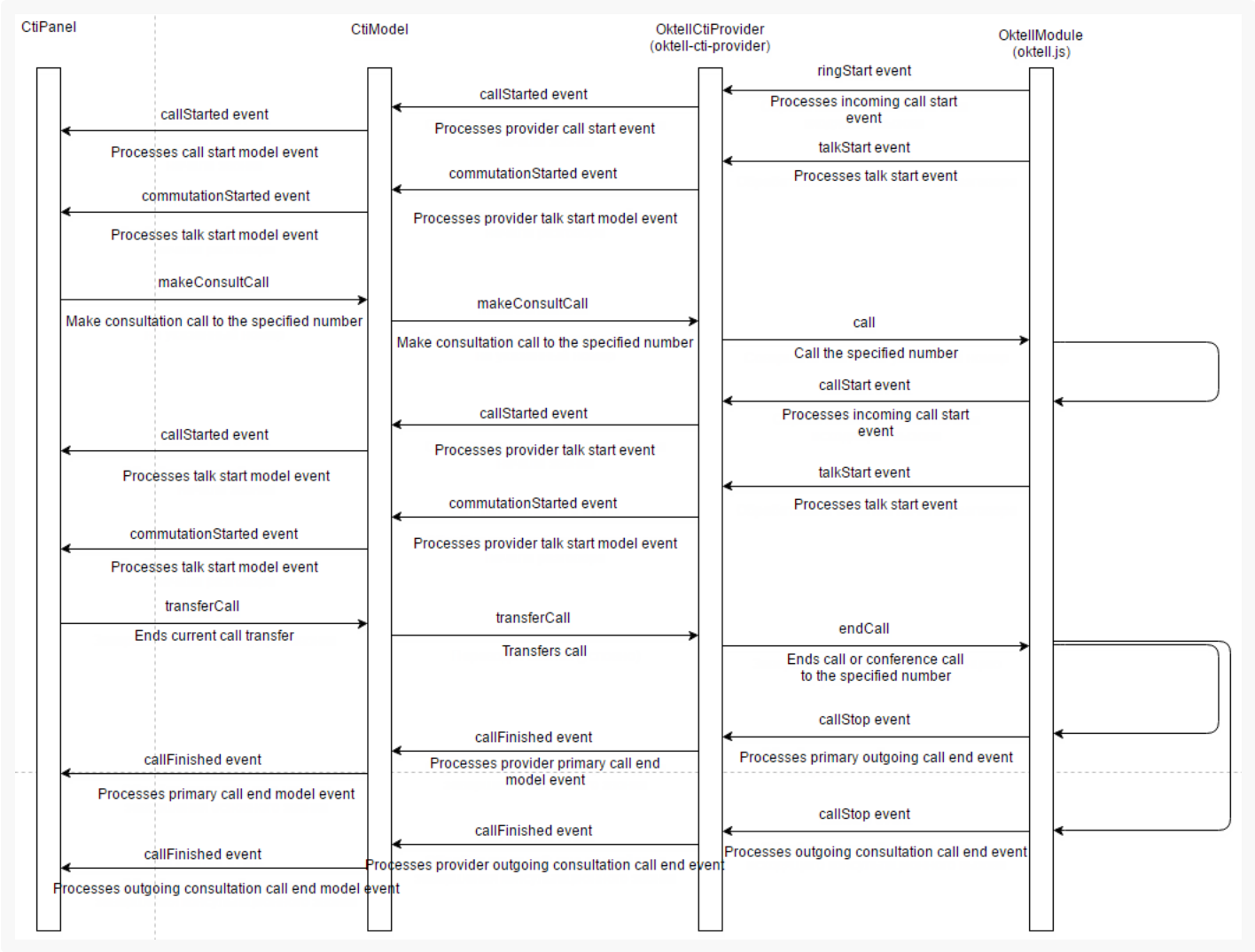


Examples of interaction between CtiModel, OktellCtiProvider and OktlellModule:

Operator outgoing call to a subscriber: putting a call on hold, putting off hold by a subscriber and finishing the call by the operator



Incoming call of a subscriber 1 to an operator with a consultation call to subscriber 2 with the subsequent connection of the subscriber 1 and subscriber 2 by the operator



The list of supported oktell.js class library events is listed in table.

The list of supported oktell.js class library events

Event	Description
`connect`	Successful connection to server event.
`connectError`	Connection to server error in the `connect` method event. Error codes are the same as for the callback function of the `connect` method.
`disconnect`	Server connection closing event. The object describing the reason of the disconnection is passed to the callback function.
`statusChange`	Agent status change event. Two string parameters are passed to the callback function — the new and previous state.
`ringStart`	Incoming call start event.
`ringStop`	Incoming call stop event.
`backRingStart`	Returning call start event.
`backRingStop`	Returning call stop event.
`callStart`	Outgoing call start event.
`callStop`	Call UUID change event.
`talkStart`	Conversation start event.
`talkStop`	Conversation stop event.
`holdAbonentLeave`	Caller hold leave event The `abonent` object is passed to the callback function with information on the caller.
`holdAbonentEnter`	Caller hold enter event The `abonent` object is passed to the callback function with information on the caller.
`holdStateChange`	Hold status change event. The information on the hold is passed to the hold function.
`stateChange`	Line status change event.
`abonentsChange`	Current abonents list change event.
`flashstatechanged`	Hold status change low-level event.
`userstatechanged`	User status change low-level event.

Integration with Webitel

Advanced

Webitel integration is implemented in the form of separate Creatio modules. Modules in the integration include:

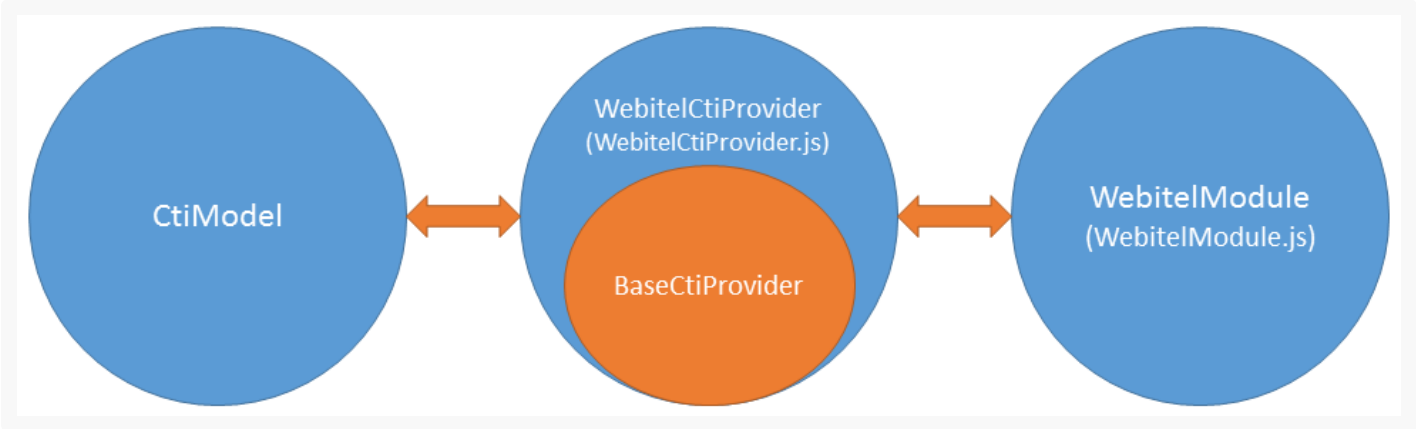
The `WebitelCore` package — modules that contain low-level interactions with Webitel using the Verto module and the CTI panel on the Creatio application page.

The `WebitelCollaborations` package implements basic interfaces for working with Webitel in Creatio. The package contains the `WebitelCtiProvider` module, the `WebitelCtiProvider` class, Webitel connector, the connection parameters settings page, the lookup to edit Webitel users directly in Creatio.

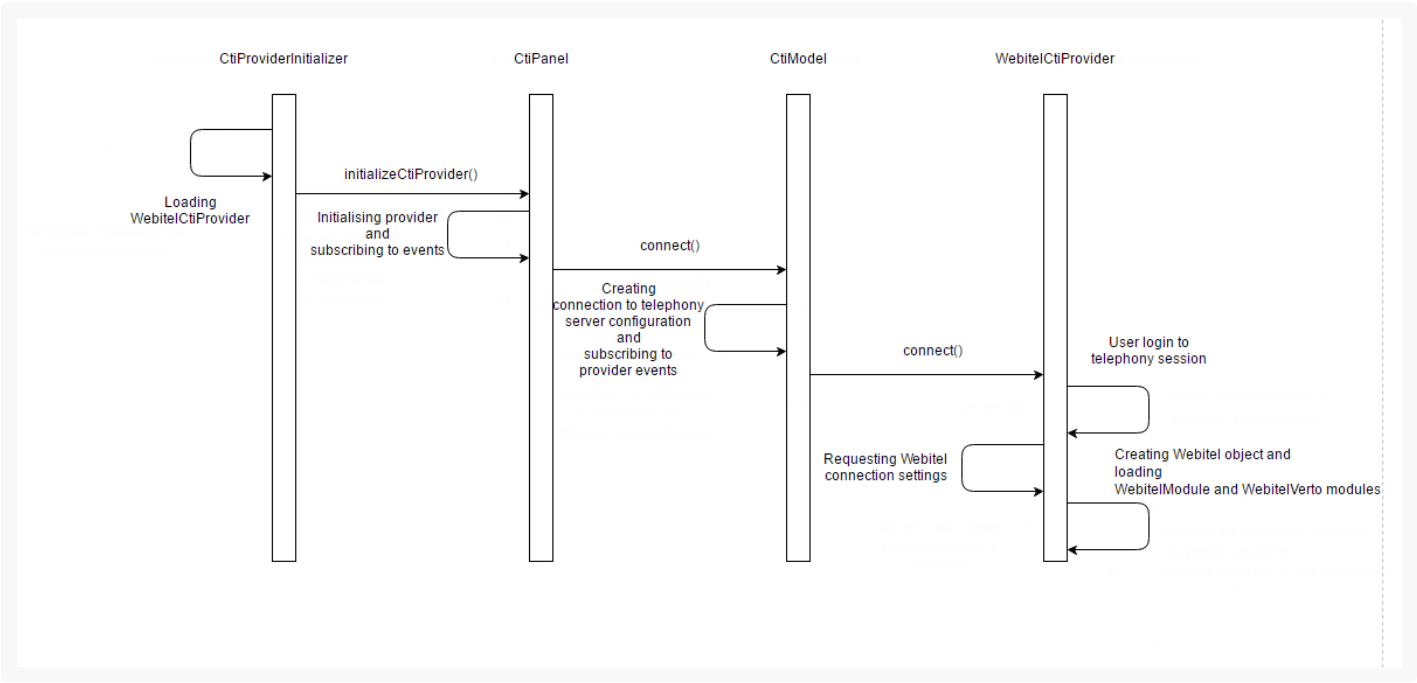
Detailed information about Webitel architecture can be found in the documentation.

Interaction of components

The `WebitelCtiProvider` class (the heir of the `Terrasoft.BaseCtiProvider` class) implements the required interaction between CtiModel and the Webitel low-level global object (the *WebitelCore.WebitelModule.js* module).



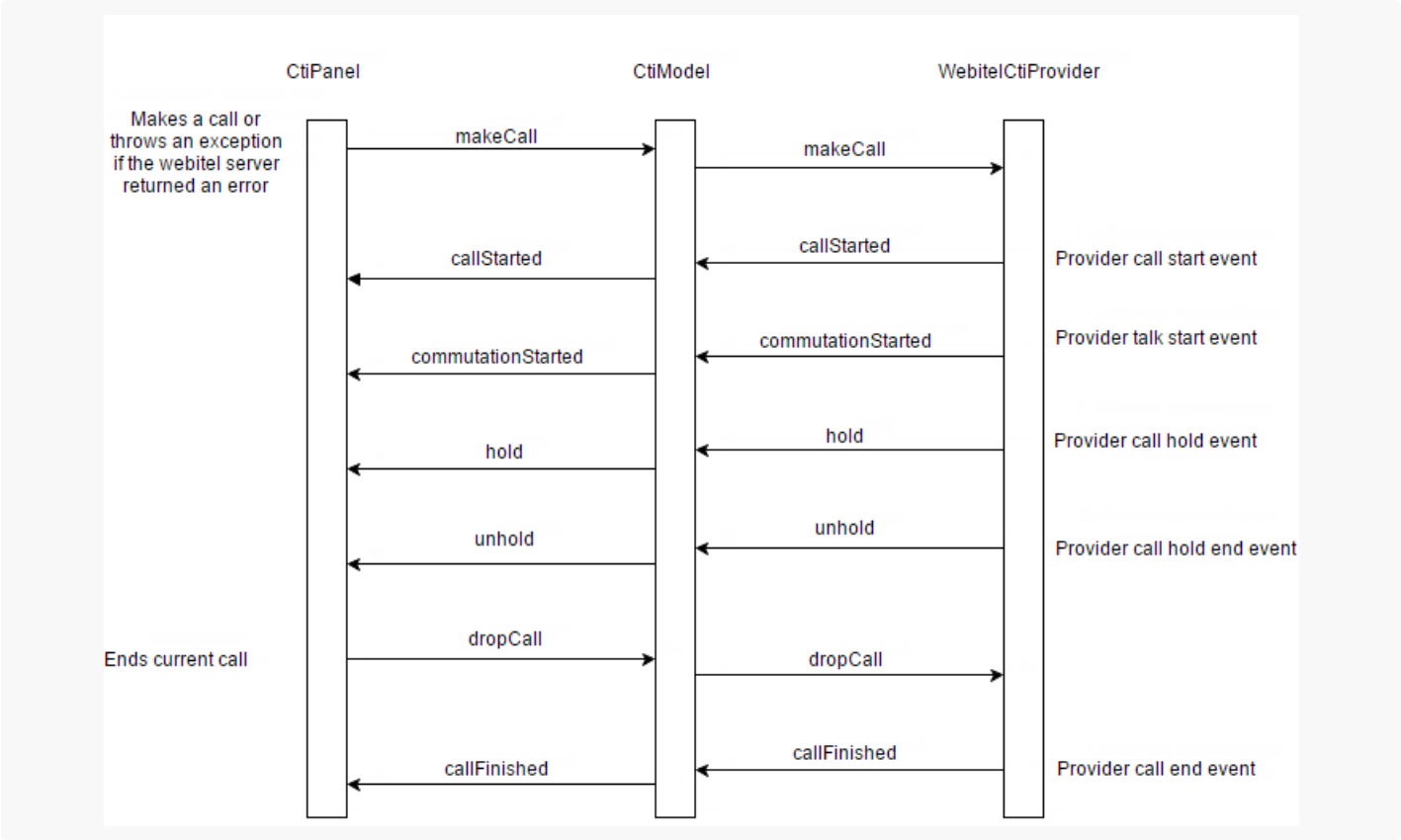
The integration is as follows. If a user has set the system setting of the Webitel integration library, *CtiProviderInitializer* loads the `WebitelCtiProvider` module. Next, it calls the `init` method in `WebitelCtiProvider`, which carries out the user login in the telephony session (the `LogInMsgServer` of the `MsgUtilService.svc` service). If the login was successful, the `connect` method is invoked, which verifies that you don't have an existing connection (the `this.isConnected` property is set to `false` and `this.webitel` — to empty). After that, the `connect` method requests the connection settings to Webitel that are stored in the system settings of the *webitelConnectionString* and *webitelWebrtcConnectionString*.



After receiving the system settings, the user settings are received from the [*Webitel users*] lookup by using the `GetUserConnection` method of the *WUserConnectionService* customer service. After receiving the user settings, the *WebitelModule* and *WebitelVerto* are loaded if the [*Use web phone*] checkbox is selected in the user settings. Next, the `onConnected` method is called that creates the `Webitel` global object, in which properties are populated with the connection settings. The subscription to `Webitel` object events occurs and the `connect` method is invoked, which performs connection via *WebSocket*, authorization of Webitel and other low-level connection operations. When the *onConnect* event occurs, the connection is considered successful and the user can work with calls. During the connector operation, `WebitelCtiProvider` reacts to `Webitel` object events, processes them, and optionally generates connector events described in the `Terrasoft.BaseCtiProvider` class. To manage calls, `WebitelCtiProvider` implements abstract methods of the `Terrasoft.BaseCtiProvider` class by using the `Webitel` object methods.

Examples of `CtiPanel`, `CtiModel` and `WebitelCtiProvider` interaction

Outgoing call to a subscriber: putting a call on hold, taking a call off hold by a subscriber and finishing a call.



Webitel list of ports

- 871 — the WebSocket port for the Webitel server and receiving events.
- 5060 and 5080 — signal ports for SIP phones and telephony providers.
- 5066 — the port for the Web phone and WebRTC signal port.
- 4004 — the port for receiving call records.

Webitel events

Event	Description
`onNewCall`	New call start event.
`onAcceptCall`	Accept call event.
`onHoldCall`	Call hold event.
`onUnholdCall`	Call Unhold event.
`onDtmfCall`	Tone dialing event.
`onBridgeCall`	Connection to channel event.
`onUuidCall`	Call UUID change event.
`onHangupCall`	Call stop event.
`onNewWebRTCCall`	New WebRTC session event.

Integration with Asterisk

Advanced

Use AMI (Asterisk Manager Interface) to interact with the Asterisk server. The API enables client programs to connect to Asterisk server by using TCP/IP protocol. The Application Programming Interface enables you to process events in the digital multiplex system (DMS), and send commands to control calls.

Note. Currently the integration of Creatio with Asterisk is supported up to version 13.

A client uses a simple text protocol for communication between the Asterisk and the connected Manager API: "parameter: value". The end of a string is determined by the sequence of Carriage Return and Line Feed (CRLF).

Note. In the future, for a set of strings like "parameter: value", followed by a blank line containing only a CRLF, for simplicity the term "package" will be used.

Set up the configuration file of the Messaging Service to integrate Asterisk to Creatio

For integration to work with Creatio, you need to install Terrasoft Messaging Service (TMS) on a dedicated computer that will be used as the integration server. You must set the following parameters for Asterisk in the `Terrasoft.Messaging.Service.exe.config` configuration file:

Parameters for Asterisk

```
<asterisk filePath="" url="Name_or_address_of_Asterisk_server" port="Asterisk_server_port"
    userName="Asterisk login" secret="Asterisk password" originateContext="Originate context" pa
    autoPauseOnCommutationStart="true" queueExtensionFormat="Local/{0}@from-queue/n" asyncOrigir
    traceQueuesState="false" packetInfoConfig="Additional package parameters to be processed in
```

Ports for Asterisk integration with Creatio

- TMS accepts WebSocket connection to the 2013 port via TCP.
- TMS connects to the Asterisk server by default via the 5038 port.

The Terrasoft Messaging Service for Asterisk integration with Creatio

The integration part of the Messaging Service is implemented in the main Creatio solution kernel in the `Terrasoft.Messaging.Asterisk` library.

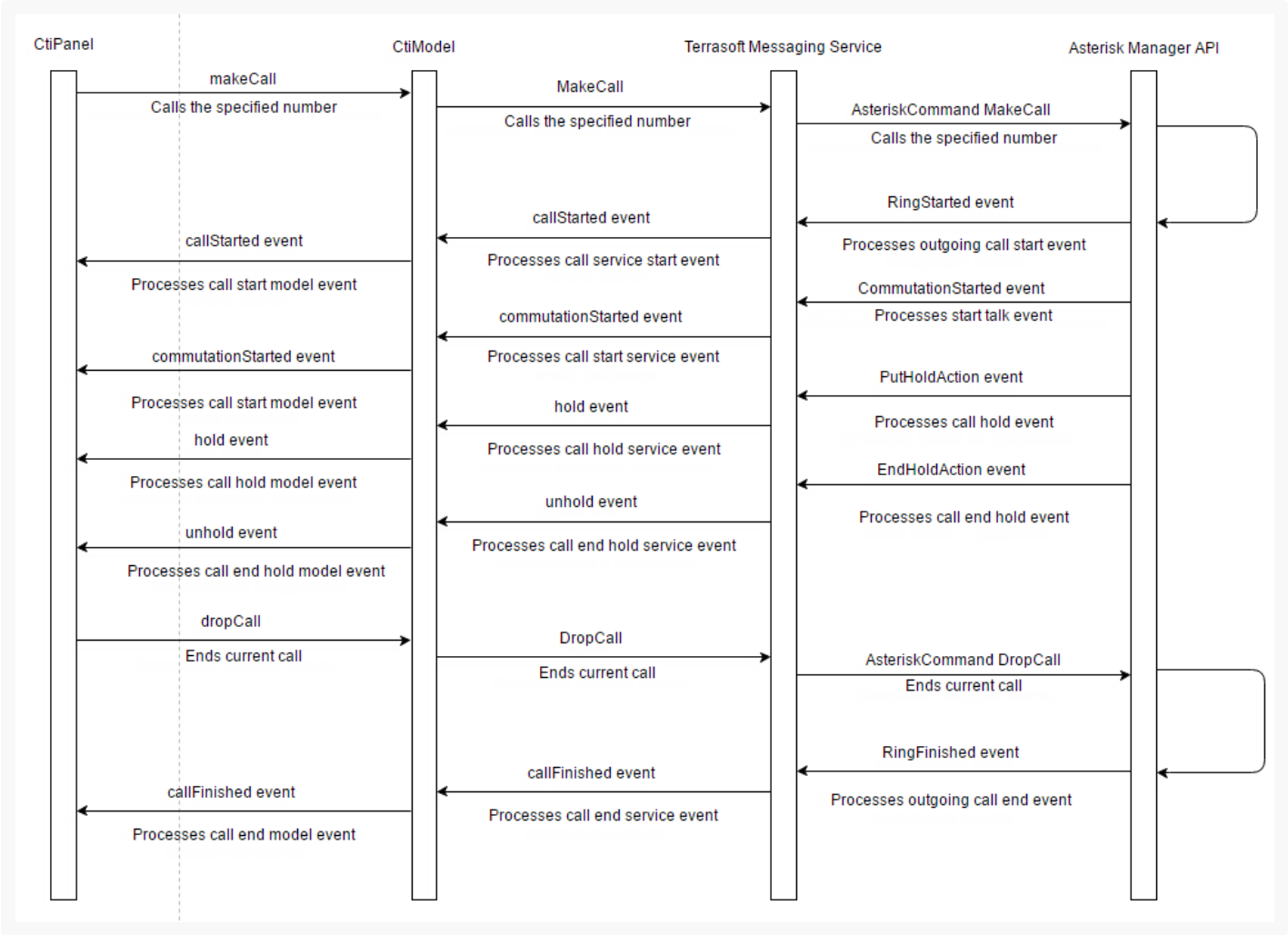
Library main classes:

- `AsteriskAdapter` — an Asterisk class that transforms events to the top-level call model events used in Creatio integration.
- `AsteriskManager` — a class that creates and deletes user connections to the Asterisk server.
- `AsteriskConnection` — a class that represents the user connection for integration with Asterisk.
- `AsteriskClient` — a class used to send commands to the Asterisk server.

Example of CtiModel, Terrasoft Messaging Service and Asterisk Manager API interaction

Operator outgoing call to a subscriber: putting a call on hold, putting off hold by a subscriber and finishing the call by the operator.

The order of events during a call for the current example:



The table shows an example of event processing including how the event data is interpreted by TMS and which values from the listed events are used when processing an incoming call.

Asterisk log	TMS	Action

`{`
`Event: newchannel`
`Channel: <channel_name>`
`UniqueID: <unique_id>`
`}`
	A channel is created and added to a collection
	`new AsteriskChannel({`
	`Name: <channel_name>,`
	`UniqueId: <unique_id>`
	`});`	

`{`
`Event: Hold`
`UniqueID: <unique_id>`
`Status: "Off"`
	Search for the channel by `<unique_id>` and generate an event by using the `fireEvent` method.	PutHoldAction

Asterisk log	TMS	Action
`{ Event: Hangup UniqueID: <unique_id> }`	Search for the channel by `<unique_id>` and generate an event by using the `fireEvent` method.	EndHoldAction
`{ Event: Dial SubEvent: Begin UniqueID: <unique_id> }`	Search for the channel by `<unique_id>` and generate an event by using the `fireEvent` method.	RingFinished
`{ Event: Dial SubEvent: Begin UniqueID: <unique_id> }`	Search for the channel by <unique_id>, fill in the data and generate an event by using the *fireEvent* method.	RingStarted
`{ Event: Bridge UniqueID: <unique_id> }`	Search for the channel by `<unique_id>` and generate an event by using the `fireEvent` method.	CommutationStarted

Asterisk events

A detail list of events and information about their parameters is described in the Asterisk documentation.