INWISE

Be ahead. Get connected!

Instrumentação do NEON via WebSocket

Especificação dos serviços de instrumentação do NEON através de conexões WebSocket.

Sumário

Instrumentação do NEON via WebSocket

Com o intuito de prover a integração com aplicativos, páginas web, módulos e componentes de software o NEON pode ser instrumentado de forma simples através de WebSocket, comportando-se como um aplicativo servidor apto a executar comandos e notificar eventos.

Através de uma conexão básica WebSocket é possível:

- Conectar e desconectar contas SIP;
- Realizar, atender e cancelar ligações;
- Receber notificações de novas ligações, atendidas e finalizadas;
- Transferir ligações;
- Colocar chamada em espera;
- Enviar DTFM;
- Apresentar e esconder interface;

Logo após a inicialização, o NEON já está apto para receber conexões de WebSocket (na porta 15070) compatível com as recomendações da W3C para comunicação com socket (https://www.w3.org/TR/websockets/).

Segue neste documento a descrição para conexão, autenticação, execução de comandos, recebimento de eventos e especificação dos formatos de dados. Todos os exemplos citados podem ser testados em qualquer navegador web compatível com HTML5.

Conexão

Para realizar qualquer iteração para instrumentação do NEON, primeiramente é necessário estabelecer a conexão utilizando o objeto de WebSocket (atualmente, padrão em todos navegadores compatível com HTML5).

Exemplo

Conexão com o NEON

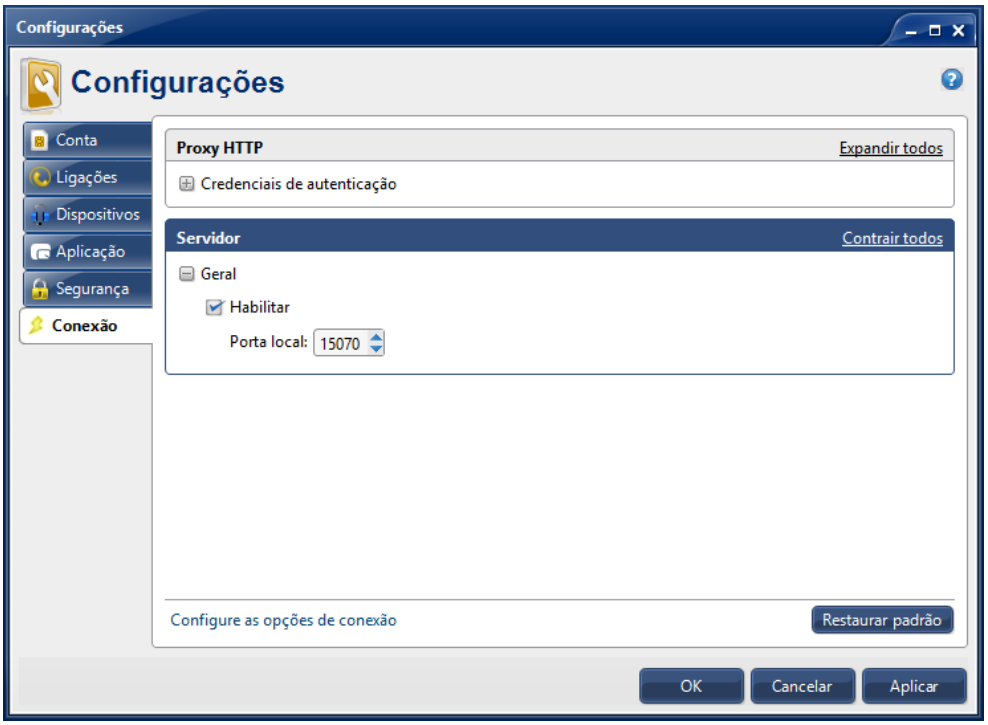
```
var websocket = new WebSocket( "ws://localhost:15070/" );

websocket.onopen = function() {
  console.log( "NEON - Connected" );
};

websocket.onclose = function() {
  console.log( "NEON - Disconnected" );
};

websocket.onerror = function( evt ) {
  console.log( "NEON - Connection error", evt );
};
```

Antes de tentar conectar-se ao NEON, é importante verificar se o servidor está habilitado para receber conexões WebSocket (conforme a imagem abaixo). Esta opção pode ser habilitada por padrão utilizando a ferramenta do NEON SlipStreamer (verificar documentação do NEON SlipStreamer).



Enviando comandos

Após conexão com sucesso, para enviar comandos é necessário preparar uma mensagem no formado JSON e utilizar o método send() do objeto de WebSocket.

Todos os comandos enviados devem ter uma propriedade action indicando qual a ação que será executada e o token de acesso, exceto o comando a*uthenticate*.

Opcionalmente, pode-se enviar uma propriedade id (identificador exclusivo único) gerado localmente para relacionar requisições as respostas recebidas no evento onmessage do objeto de WebSocket.

Exemplo

Enviando comando

```
websocket.send( JSON.stringify({

  id: 1,
  token: "24b806ff",
  action: "startcall",
  number: "7155555555"

}));
```

Resposta

```
websocket.onmessage = function( evt ) {

  console.log( evt.data );
  var data = JSON.parse( evt.data );

  // Ex: output do objeto data
  // {
  //  id: 1,
  //  success: true,
  //  data:
  //  {
  //    callId: 55
  //  }
  // }
  //

}
```

Escutando eventos

Através do evento de onmessage do objeto de WebSocket, é possível tratar os eventos gerados no NEON. Todos os eventos gerados no NEON serão enviados como uma mensagem no formato JSON e poderá ser identificado através das propriedades *action* e *name*.

Exemplo

Escutando eventos

```
websocket.onmessage = function( evt ) {

    var data = JSON.parse( evt.data );
    if ( data.action == "event" )
        console.log( "NEON - Event: ", data );

    // Ex: output do objeto data para um evento de ligação
    //{
    //    "action": "event",
    //    "name": "OnNewCall",
    //    "data": {
    //        "callId": 55,
    //        "type": "outgoing",
    //        "from": "sip:20719023@208.89.104.57",
    //        "to": "sip:2342342@208.89.104.57"
    //    }
    //}

}
```

Cada tipo de evento, conforme a natureza do evento, enviará propriedades adicionais (verificar sessão referência deste documento).

Observação: Respostas de comandos enviados e eventos gerados no NEON são recebidos no mesmo evento de onmessage do objeto WebSocket, portanto é importante testar o valor da propriedade *action* para tratar eventos. Para saber o tipo do evento, basta verificar o valor da propriedade *name*.

Autenticação

Para obter acesso a API de instrumentação e receber eventos, primeiramente, deve-se enviar o comando de autenticação identificando a aplicação que instrumentará o NEON com um nome de usuário e senha.

O comando *authenticate* retornará um token de acesso que deverá ser utilizado para execução de qualquer outro comando. Este é o único comando que não necessita de um token de acesso.

Exemplo

Requisição

```
{
  id: 0,
  action: "authenticate",
  user: "[appUserId]",
  password: "[appUserPassword]"
}
```

Resposta

```
{
  id: 0,
  success: true,
  data:
  {
    token: "24b806ff"
  }
}
```

Conectado conta SIP

Após autenticação, para conectar uma conta SIP no NEON é necessário enviar o comando *SIPRegister*. Além do nome do usuário SIP e senha, opcionalmente (através do parâmetro *autodisconnect*), é possível indicar que a conta SIP deve ser desconectada automaticamente caso a conexão com o WebSocket seja finalizada.

Após execução do *SIPRegister* o NEON poderá gerar os eventos *OnSIPRegister, OnSIPRegisterError, OnSIPRegisterCanceled, OnSIPRegisterFinalized, OnSIPUnregister.*

Para desconectar uma conta, basta enviar o comando *SIPUnRegister.* Verificar a sessão de referência abaixo para maiores detalhes sobre estes comandos e eventos.

Referência

Comandos

Nome	Parâmetros
Authenticate: Realiza autenticação da aplicação cliente que instrumentará o NEON	action: authenticate user: *nome da aplicação cliente* password: *senha*
`Ex: { action:"authenticate", user:"xxxx", password:"xxxx" }`	
SIPRegister: Conecta conta SIP	domain: Endereço:porta do sip proxy number: *Número/Nome do susuário* password: *Senha* autodisconnect: *Valor booleano indicando se deve desconectar ao perder conexão WebSocket* token: *Chave de acesso da aplicação do cliente*
`Ex: { action:"sipregister", domain: "xxxx", number:"xxxx", password:"xxxx", autodisconnect:true, token:"xxxx" }`	
SIPUnRegister: Desconecta conta SIP	token: *Chave de acesso da aplicação do cliente*
`Ex: { action:"sipunregister", token:"xxxx" }`	
StartCall: Realiza uma nova ligação para um determinado número	action: startcall number: *Número a ser discado* token: *Chave de acesso da aplicação do cliente*
`Ex: { action:"startcall", number:"xxxx", token:"xxxx" }`	
FinalizeCall: Finaliza uma ligação através do Id da chamada recebido em um dos eventos de ligação	action: finalizecall callid: *Id da ligação que deseja finalizar* token: *Chave de acesso da aplicação do cliente*
`Ex: { action:"finalizecall", callid:"xxxx", token:"xxxx" }`	

AcceptCall: Atende uma ligação recebida através do Id da chamada	action: acceptcall callid: *Id da ligação que deseja atender* token: *Chave de acesso da aplicação do cliente*
`Ex: { action:"acceptcall", callid:"xxxx", token:"xxxx" }`	
TransferCall: Transfere uma ligação em recebimento ou atendida para determinado número	action: transfercall callid: *Id da ligação que deseja transferir* to: *Número que deseja transferir a ligação* token: *Chave de acesso da aplicação do cliente*
`Ex: { action:"transfercall", callid:"xxxx", to:"xxxx", token:"xxxx" }`	
SendDTMF: Envia sinal de DTMF em uma ligação em andamento	action: senddtmf callid: *Id da ligação que deseja enviar DTMF* DTMF: *Número correspondente ao digito DTMF* token: *Chave de acesso da aplicação do cliente*
`Ex: { action:"senddtmf", callid:"xxxx", DTMF:"xxxx", token:"xxxx" }`	
SetHold: Coloca uma ligação em estado de espera	action: sethold callid: *Id da ligação* hold: *Valor booleano indicando estado de espera* token: *Chave de acesso da aplicação do cliente*
`Ex: { action:"sethold", callid:"xxxx", hold:true, token:"xxxx" }`	
ShowUI: Exibe ou esconde a interface do NEON	show: *Valor booleano indicando se deve exibir ou esconder*
`Ex: { action:"showui", show:true, token:"xxxx" }`	

Eventos

Nome	Dados
OnSIPRegister: Indica que uma conta SIP conectou com sucesso.	`connectionId: Id da conexão SIP` `SIPError: Código de erro SIP`
OnSIPUnRegister: Indica que uma conta SIP desconectou com sucesso.	
OnSIPRegisterRefused: Indica o processo de registro da conta SIP foi recusado pelo servidor.	
OnSIPRegisterFinalized: Indica que o processo de registro foi finalizado.	
OnSIPRegisterCancelled: Indica que o processo de registro foi cancelado.	
OnSIPRegisterExpired: Indica que o tempo para registro da conta SIP expirou.	

OnNewCall: Indica que uma nova ligação realizada ou recebida foi iniciada.	`callId: Id da ligação`
	`type: incoming (recebida) outgoing (realizada)`
	`from: SIP URI do originário`
	`to: SIP URI do destinatário`
	`duration: Duração da ligação em milissegundos`
	`SIPError: Código de erro SIP`

OnFinalizedCall: Indica que uma ligação foi finalizada.	
OnAcceptCall: Indica que uma ligação foi atendida.	
OnTransferedCall: Indica que uma ligação foi transferida.	

Códigos de Erro

Código	Descrição
0	Nenhum erro
100	Erro no formato dos dados
102	Argumento inválido
500	Erro interno
600	Credenciais inválidas
601	Não registrado no SIP Proxy
602	Execução do comando não autorizado

Maiores informações

Salvador, 17 de abril de 2018.
Versão do documento: 2.1
Última atualização: 17 de abril de 2018

InWise Internet Company do Brasil S.A.
www.inwise.com.br
Tel: +55 (71) 3341.1525
Email: joselito@inwise.com.br