Platform SDK Developer's Guide

Connecting Using UTF-8 Character Encoding

5/1/2025

Connecting Using UTF-8 Character Encoding

Java

Scenarios

Genesys Configuration Server 8.1.2 added the ability to be configured to support multiple languages at a same time using UTF-8 encoding. Once Configuration Server is installed, configured and started in multilingual (UTF-8) mode it cannot be switched to regular mode. If Configuration Server is installed and started in normal mode, then it cannot be switched to multilingual (UTF-8) mode later.

One known issue is that the UTF-enabled protocol breaks backward compatibility, so users must add their own code for connection reconfiguration. The following samples describe connection scenarios with Platform SDK:

Scenario 1

Configuration Server is release 8.1.2 or later and is NOT configured as multilingual (without UTF-8 transport), or is an earlier version without support for the UTF-8 feature.

In this scenario, Platform SDK connections can be created in the usual way.

Scenario 2

Configuration Server is release 8.1.2 or later and configured as multilingual (with UTF-8 transport), with:

A) Platform SDK release 8.1.3 in use.

Reconfiguration for encoding is automatically handled by Platform SDK as described in the section below – no user action is required.

B) Platform SDK release 8.1.1 or 8.1.2 in use.

Platform SDK provides information that Configuration Server is UTF-8, so, the connection can be reopened using new connection configuration with following user code.

```
PropertyConfiguration config = new PropertyConfiguration();
config.setUseAddp(true);
config.setAddpClientTimeout(11);
config.setAddpServerTimeout(21);

ConfServerProtocol protocol = new ConfServerProtocol(new Endpoint(name, host, port, config));
```

```
protocol.setClientName(clientName);
protocol.setClientApplicationType(clientType.ordinal());
protocol.setUserName(username);
protocol.setUserPassword(password);

protocol.open();

Integer cfgServerEncoding = protocol.getServerContext().getServerEncoding();
if (cfgServerEncoding != null && cfgServerEncoding.intValue() == 1) {
   protocol.close();
   config.setStringsEncoding("UTF-8");
   protocol.setEndpoint(new Endpoint(name, host, port, config));
   protocol.open();
 }
```

It may be more comfortable to move the flag value evaluation to a separated method where a temporary `ConfServerProtocol` instance may be created - especially in the case of `ChannelListeners` usage, messages handlers, etc.

> **Important**
>
> This is not the best solution for wide usage. The `ServerEncoding` value evaluation method may fail if non-ASCII symbols are found inside the username or password, which may lead to a handshake procedure error such as "invalid username/password". This issue may be resolved with an additional test connection retry with UTF-8 enabled, but this workaround is not a best practice solution.

C) Platform SDK release 8.0.1 through 8.1.1 in use:

Platform SDK does NOT indicate whether Configuration Server is using UTF-8 mode or not, so user application should take care to evaluate this information (or have it defined by the design or configuration of the application).

In this case we have no `protocol.getServerContext().getServerEncoding()`, but we are able to configure the connection for Unicode usage.

It may be recommended to add one more property to the application configuration/parameters (along with the existing Configuration Server host and port) such as a boolean "isCSUTF8" value.

```
PropertyConfiguration config = new PropertyConfiguration();
if (isCSUTF8) {
   config.setOption(Connection.STR_ATTR_ENCODING_NAME_KEY, "UTF-8");
}

ConfServerProtocol protocol = new ConfServerProtocol(new Endpoint(name, host, port, config));
protocol.setClientName(clientName);
protocol.setClientApplicationType(clientType.ordinal());
protocol.setUserName(username);
protocol.setUserPassword(password);

protocol.open();
```

D) Platform SDK release of 8.0.0 or earlier in use.

No support if provided for string encoding of connection configuration options. The only way is to use this feature is to upgrade your release of Platform SDK.

Automatic UTF-8 Character Encoding Set Up on Handshake

Starting in Platform SDK Release 8.1.3 (which incorporates Configuration Protocol Release 3.79), support for UTF-8 encoding can be automatically detected.

The process for this features is described here:

1. The first handshake message, `EventProtocolVersion`, now includes the extra `ServerEncoding` attribute. If this attribute is 1 then Platform SDK updates string encoding for that connection to the server as UTF-8.
2. The next message from the client requests authentication from the server. These messages (`RequestRegisterClient` or `RequestRegisterClient2`) have been expanded with the `ClientEncoding` attribute, which must have the same value as the `ServerEncoding` attribute received previously.
3. After the handshake is complete, string encoding for this channel may be different from the string encoding specified in the original configuration parameters. You can access the current value through `Endpoint.GetConfiguration()` of the `ConfServerProtocol` instance.

.NET

Scenarios

Genesys Configuration Server 8.1.2 added the ability to be configured to support multiple languages at a same time using UTF-8 encoding. Once Configuration Server is installed, configured and started in multilingual (UTF-8) mode it cannot be switched to regular mode. If Configuration Server is installed and started in normal mode, then it cannot be switched to multilingual (UTF-8) mode later.

One known issue is that the UTF-enabled protocol breaks backward compatibility, so users must add their own code for connection reconfiguration. The following samples describe connection scenarios with Platform SDK:

Scenario 1

Configuration Server is release 8.1.2 or later and is NOT configured as multilingual (without UTF-8 transport), or is an earlier version without support for the UTF-8 feature.

In this scenario, Platform SDK connections can be created in the usual way.

Scenario 2

Configuration Server is release 8.1.2 or later and configured as multilingual (with UTF-8 transport),

with:

A) Platform SDK release 8.1.3 in use.

Reconfiguration for encoding is automatically handled by Platform SDK as described in the section below – no user action is required.

B) Platform SDK release 8.1.1 or 8.1.2 in use.

Platform SDK provides information that Configuration Server is UTF-8, so, the connection can be reopened using new connection configuration with following user code.

```
PropertyConfiguration config = new PropertyConfiguration();
config.UseAddp = true;
config.AddpClientTimeout = 11;
config.AddpServerTimeout = 21;

ConfServerProtocol protocol = new ConfServerProtocol(new Endpoint(_name, _host, _port,
config));
protocol.ClientName = _clientName;
protocol.ClientApplicationType = _clientType;
protocol.UserName = _userName;
protocol.UserPassword = _password;

protocol.Open();

int? cfgServerEncoding = protocol.Context.ServerEncoding;
if (cfgServerEncoding != null && cfgServerEncoding.Value == 1)
{
  protocol.Close();
  config.StringsEncoding = "UTF-8";
  protocol.Endpoint = new Endpoint(_name, _host, _port, config);
  protocol.Open();
}
```

It may be more comfortable to move the flag value evaluation to a separated method where a temporary `ConfServerProtocol` instance may be created - especially in the case of `ChannelListeners` usage, messages handlers, etc.

> **Important**
>
> This is not the best solution for wide usage. The `ServerEncoding` value evaluation method may fail if non-ASCII symbols are found inside the username or password, which may lead to a handshake procedure error such as "invalid username/password". This issue may be resolved with an additional test connection retry with UTF-8 enabled, but this workaround is not a best practice solution.

C) Platform SDK release 8.0.1 through 8.1.1 in use:

Platform SDK does NOT indicate whether Configuration Server is using UTF-8 mode or not, so user application should take care to evaluate this information (or have it defined by the design or configuration of the application).

In this case we have no `protocol.Context.ServerEncoding` property, but we are able to configure

the connection for Unicode usage.

It may be recommended to add one more property to the application configuration/parameters (along with the existing Configuration Server host and port) such as a boolean "isCSUTF8" value.

```
PropertyConfiguration config = new PropertyConfiguration();
if (_isCsutf8)
{
  config.SetOption(CommonConnection.StringAttributeEncodingKey, "UTF-8");
}

ConfServerProtocol protocol = new ConfServerProtocol(new Endpoint(_name, _host, _port,
config));
protocol.ClientName = _clientName;
protocol.ClientApplicationType = _clientType;
protocol.UserName = _userName;
protocol.UserPassword = _password;

protocol.Open();
```

D) Platform SDK release of 8.0.0 or earlier in use.

No support if provided for string encoding of connection configuration options. The only way is to use this feature is to upgrade your release of Platform SDK.

Automatic UTF-8 Character Encoding Set Up on Handshake

Starting in Platform SDK Release 8.1.3 (which incorporates Configuration Protocol Release 3.79), support for UTF-8 encoding can be automatically detected.

The process for this features is described here:

1. The first handshake message, `EventProtocolVersion`, now includes the extra `ServerEncoding` attribute. If this attribute is 1 then Platform SDK updates string encoding for that connection to the server as UTF-8.
2. The next message from the client requests authentication from the server. These messages (`RequestRegisterClient` or `RequestRegisterClient2`) have been expanded with the `ClientEncoding` attribute, which must have the same value as the `ServerEncoding` attribute received previously.
3. After the handshake is complete, string encoding for this channel may be different from the string encoding specified in the original configuration parameters. You can access the current value through `Endpoint.GetConfiguration()` of the `ConfServerProtocol` instance.