



# Using TCP Adapters in Productions

Version 2023.1  
2024-07-11

*Using TCP Adapters in Productions*

InterSystems IRIS Data Platform Version 2023.1 2024-07-11

Copyright © 2024 InterSystems Corporation

All rights reserved.

InterSystems®, HealthShare Care Community®, HealthShare Unified Care Record®, IntegratedML®, InterSystems Caché®, InterSystems Ensemble®, InterSystems HealthShare®, InterSystems IRIS®, and TrakCare are registered trademarks of InterSystems Corporation. HealthShare® CMS Solution Pack™ HealthShare® Health Connect Cloud™, InterSystems IRIS for Health™, InterSystems Supply Chain Orchestrator™, and InterSystems TotalView™ For Asset Management are trademarks of InterSystems Corporation. TrakCare is a registered trademark in Australia and the European Union.

All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

This document contains trade secret and confidential information which is the property of InterSystems Corporation, One Memorial Drive, Cambridge, MA 02142, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

For Support questions about any InterSystems products, contact:

**InterSystems Worldwide Response Center (WRC)**

Tel: +1-617-621-0700

Tel: +44 (0) 844 854 2917

Email: [support@InterSystems.com](mailto:support@InterSystems.com)

# Table of Contents

<b>1 Using the Inbound TCP Adapters .....</b>	<b>1</b>
1.1 Overview of Inbound TCP Adapter .....	1
1.2 Overall Behavior .....	1
1.3 Creating a Business Service to Use a TCP Inbound Adapter .....	2
1.4 Implementing the OnProcessInput() Method .....	3
1.4.1 Signature for OnProcessInput() for EnsLib.TCP.CountedInboundAdapter .....	3
1.4.2 Signature for OnProcessInput() for EnsLib.TCP.CountedXMLInboundAdapter .....	4
1.4.3 Signature for OnProcessInput() for EnsLib.TCP.TextLineInboundAdapter .....	4
1.4.4 Implementing OnProcessInput() .....	4
1.5 Example for EnsLib.TCP.TextLineInboundAdapter .....	5
1.6 Adding and Configuring the Business Service .....	5
1.7 Related Options .....	6
<b>2 Using the Outbound TCP Adapters .....</b>	<b>7</b>
2.1 Overview of Outbound TCP Adapter .....	7
2.2 Overall Behavior .....	7
2.3 Creating a Business Operation to Use a TCP Outbound Adapter .....	7
2.4 Creating Message Handler Methods .....	9
2.4.1 Calling Adapter Methods from the Business Operation .....	9
2.5 Example for EnsLib.TCP.CountedOutboundAdapter .....	10
2.6 Adding and Configuring the Business Operation .....	11
2.7 Related Options .....	11
<b>3 Special Topics for TCP Adapters .....</b>	<b>13</b>
3.1 Adding a TCP Status Service .....	13
3.2 Creating Custom TCP Adapter Classes .....	14
3.3 Common Customizations in TCP Adapter Subclasses .....	15
<b>TCP Adapter Settings .....</b>	<b>17</b>
Settings for the TCP Inbound Adapters .....	18
Settings for the TCP Outbound Adapters .....	22



# 1

## Using the Inbound TCP Adapters

This page describes how to use the primary TCP inbound adapters (`EnsLib.TCP.CountedInboundAdapter`, `EnsLib.TCP.CountedXMLInboundAdapter`, and `EnsLib.TCP.TextLineInboundAdapter`).

### 1.1 Overview of Inbound TCP Adapter

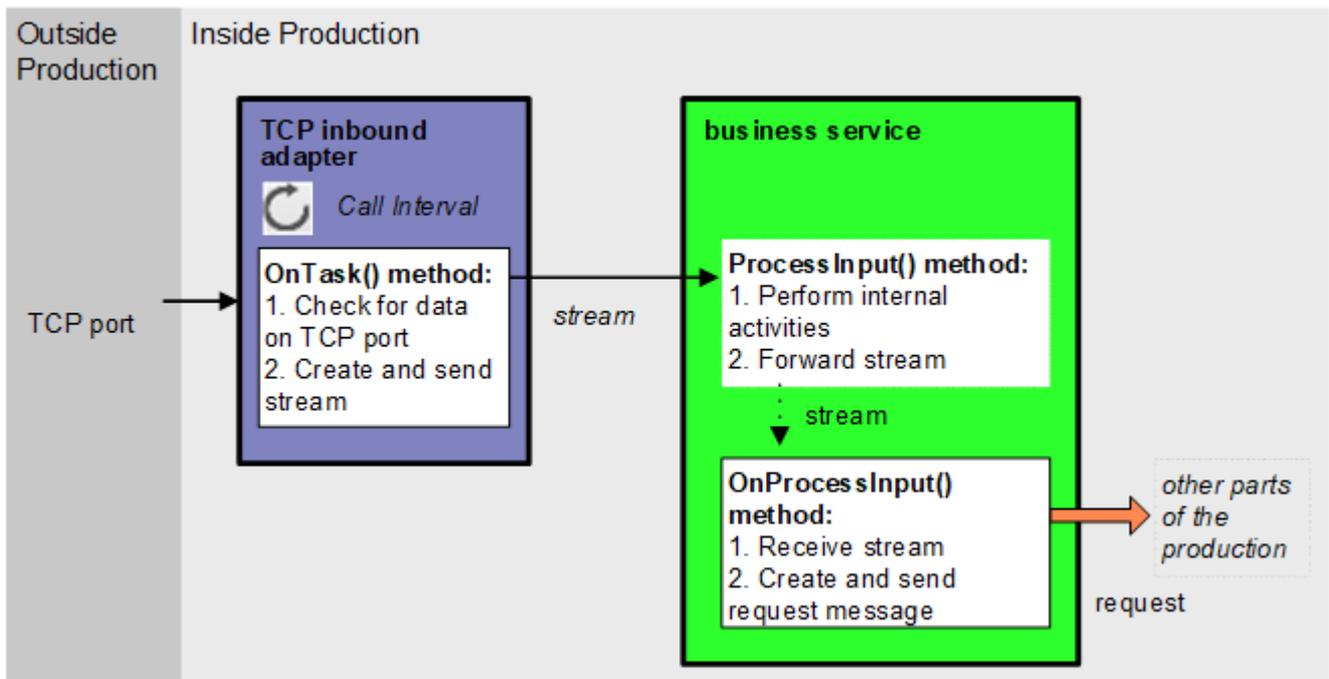
InterSystems IRIS® data platform provides the following inbound TCP adapters, all of which are subclasses of `EnsLib.TCP.InboundAdapter`:

- `EnsLib.TCP.CountedInboundAdapter` supports incoming TCP connections over which a TCP client and TCP listener exchange blocks of data, with the block length specified in the first 4 bytes of the block. The adapter uses the block length to acquire the meaningful portion of the data from the client application.
- `EnsLib.TCP.CountedXMLInboundAdapter` acts as a TCP listener for an XTE server. The adapter receives XML data in counted TCP format and imports the XML data into InterSystems IRIS. The result is an instantiated InterSystems IRIS object.
- `EnsLib.TCP.TextLineInboundAdapter` supports incoming TCP connections over which a TCP client and TCP listener exchange data as text strings that end with a line terminator character. The default terminator is the newline character (ASCII 10).

An important example of a business service that uses the `EnsLib.TCP.TextLineInboundAdapter` is the built-in business service class `EnsLib.TCP.StatusService`. This class provides a command-line interface that allows a console user or command-line script to retrieve basic status information from a running production. For more information, see [Adding a TCP Status Service](#).

### 1.2 Overall Behavior

Each TCP inbound adapter checks for data on a specified port, reads the input, and sends the input as a stream to the associated business service. The business service, which you create and configure, uses this stream and communicates with the rest of the production. The following figure shows the overall flow:



In more detail:

1. Each time the adapter encounters input from its configured data source, it calls the internal **ProcessInput()** method of the business service class, passing the stream as an input argument.
2. The internal **ProcessInput()** method of the business service class executes. This method performs basic production tasks such as maintaining internal information as needed by all business services. You do not customize or override this method, which your business service class inherits.
3. The **ProcessInput()** method then calls your custom **OnProcessInput()** method, passing the input object. The requirements for this method are described later in this page.

The response message follows the same path, in reverse.

## 1.3 Creating a Business Service to Use a TCP Inbound Adapter

To use any of the TCP adapters in your production, create a new business service class as described here. Later, [add it to your production and configure it](#).

You must also create appropriate message classes, if none yet exist. See [Defining Messages](#).

The following list describes the basic requirements of the business service class:

- Your business service class should extend `Ens.BusinessService`.
- In your class, the *ADAPTER* parameter should equal `EnsLib.TCP.CountedInboundAdapter`, `EnsLib.TCP.CountedXMLInboundAdapter`, or `EnsLib.TCP.TextLineInboundAdapter`.
- Your class must implement the `OnProcessInput()` method, as described in [Implementing the OnProcessInput Method](#).
- Your class can implement the **OnConnect()** callback method. This method is called after the **OnTask()** method, each time the TCP inbound adapter establishes a new connection to or from a remote system.

If you implement this method, it must have the following signature:

```
Method OnConnect(pTimeout As %Numeric) As %Status
```

Implement **OnConnect()** to take some action each time a new connection is established, for example, to send a logon sequence or a handshake token. The timeout argument is automatically provided by the TCP inbound adapter. It takes its value from the Read Timeout adapter setting. For details about Read Timeout, see [Reference for Settings](#).

If **OnConnect()** returns an error status, the new connection fails and the adapter is disconnected. If an untrapped exception occurs within **OnConnect()**, the adapter catches it and continues as if **OnConnect()** were not implemented.

For other options and general information, see [Defining a Business Service Class](#).

- Your class can implement the **OnInit()** method to perform any special setup actions. This method is called from the **%OnNew()** method. If the [Job Per Connection](#) setting is set to `true`, then **OnInit()** is called from **%OnNew()** each time the listener spawns a new job to accept a connection.
- Your class can implement the **OnTearDown()** method to perform any special tear down actions. This method is called from the **%OnClose()** method. Specifically, **%OnClose()** calls the **OnTearDown()** method of the adapter and then calls the **OnTearDown()** method of the business service. If the [Job Per Connection](#) setting is set to `true`, then **OnTearDown()** is called just before each connection job is closed.

**Important:** If your new business service class includes custom code in the **OnTearDown()** method, the custom code must invoke the **OnTearDown()** method of the superclass to enable the proper functioning of the business service, for example:

```
//Sets a node of the ^%zexample global to the ID of the current process
// and invokes the OnTearDown() method of the superclass
Method OnTearDown() As %Status
{
    set ^%zexample($i(^%zexample), "onteardown")=$j
    Return ##super()
}
```

You must also implement Try/Catch logic as appropriate to handle errors. For information about invoking a superclass method, see [##superclass Syntax](#).

## 1.4 Implementing the OnProcessInput() Method

This section describes the method signature for **OnProcessInput()**, which depends upon the adapter, and describes how to [implement](#) this method.

### 1.4.1 Signature for OnProcessInput() for EnsLib.TCP.CountedInboundAdapter

If your business service class uses `EnsLib.TCP.CountedInboundAdapter`, your **OnProcessInput()** method should have the following signature:

```
Method OnProcessInput(pInput As %Library.GlobalCharacterStream,
                    Output pOutput As %Library.AbstractStream) As %Status
```

Where:

- *pInput* contains the incoming data stream that the TCP client has directed to the adapter.
- *pOutput* contains any response that the business service might provide to the TCP client.

## 1.4.2 Signature for OnProcessInput() for EnsLib.TCP.CountedXMLInboundAdapter

If your business service class uses `EnsLib.TCP.CountedXMLInboundAdapter`, your **OnProcessInput()** method should have the following signature:

```
Method OnProcessInput(pInput As %RegisteredObject,  
                    Output pOutput As %RegisteredObject) As %Status
```

Where:

- *pInput* can be any of the objects specified by the Accept Class Names adapter setting. For details about Accept Class Names, see [Reference for Settings](#).
- *pOutput* contains any response that you might need to return to the XTE server.

## 1.4.3 Signature for OnProcessInput() for EnsLib.TCP.TextLineInboundAdapter

If your business service class uses `EnsLib.TCP.TextLineInboundAdapter`, your **OnProcessInput()** method should have the following signature:

```
Method OnProcessInput(pInput As Ens.StringContainer,  
                    Output pOutput As Ens.StringContainer) As %Status
```

Where:

- *pInput* contains the incoming line of text.
- *pOutput* contains the outgoing response string (if any).

## 1.4.4 Implementing OnProcessInput()

In all cases, the **OnProcessInput()** method should do some or all of the following:

1. Examine the input object (*pInput*) and decide how to use it.
2. Create an instance of the request message, which will be the message that your business service sends. For information on creating message classes, see [Defining Messages](#).
3. For the request message, set its properties as appropriate, using values in the input.
4. Call a suitable method of the business service to send the request to some destination within the production. Specifically, call **SendRequestSync()**, **SendRequestAsync()**, or (less common) **SendDeferredResponse()**. For details, see [Sending Request Messages](#). Each of these methods returns a status (specifically, an instance of `%Status`).
5. Make sure that you set the output argument (*pOutput*). Typically you set this equal to the response message that you have received. This step is required.
6. If the data source expects an acknowledgment or response to its input, **OnProcessInput()** must create this response and relay it to the data source, via the adapter.
7. Return an appropriate status. This step is required.

## 1.5 Example for EnsLib.TCP.TextLineInboundAdapter

The following is an example of a business service class that uses EnsLib.TCP.TextLineInboundAdapter.

### Class Definition

```

Class TestTCPTextLine.AuthorizationTCPService Extends Ens.BusinessService
{
  /// Name of the adapter class
  Parameter ADAPTER = "EnsLib.TCP.TextLineInboundAdapter";

  Method OnProcessInput(pInput As Ens.StringContainer,
                      pOutput As Ens.StringContainer) As %Status
  {
    set $ZT = "EXCEPTION"
    set st = $$$OK

    do {
      if (!$isobject($get(pInput))) { quit }

      // Input must have the following format: 'PatientCode:ProcedureCode'
      set tSubject = pInput.StringValue
      $$$TRACE("received line "_tSubject)

      set req = ##class(TestTCPTextLine.AuthorizationRequest).%New()
      set req.PatientCode = $piece(tSubject,":",1)
      set req.ProcedureCode = $piece(tSubject,":",2)

      set st = ..SendRequestSync("AuthorizationProcess", req, .resp)
      quit:$$$ISERR(st)

      set pOutput=
        ##class(Ens.StringContainer).%New(resp.AuthorizationFlag_
          ":",_resp.AuthorizationCode)
    } while (0)

  EXIT
  //do ..Adapter.Disconnect()
  quit st

  EXCEPTION
  set $ZT = ""
  set st = $$$EnsSystemError
  goto EXIT
}

```

## 1.6 Adding and Configuring the Business Service

To add your business service to a production, use the Management Portal to do the following:

1. Add an instance of your business service class to the production.
2. Configure the business service. For information on the settings, see [Reference for Settings](#).

When you configure the business service, you specify a value for the **Allowed IP Addresses** settings, along other settings. Note that **Allowed IP Addresses** provides a way to enable the adapter to initiate the connection.

3. Enable the business service.
4. Run the production.

## 1.7 Related Options

EnsLib.TCP contains additional TCP inbound adapters such as EnsLib.TCP.FramedInboundAdapter. See the class reference for details.

InterSystems IRIS also provides specialized business service classes that use TCP adapters, and one of those might be suitable for your needs. If so, no programming would be needed. See [Connectivity Options](#).

You can develop a new inbound adapter class based on the EnsLib.TCP.InboundAdapter or any of its subclasses. See [Creating Custom TCP Adapter Classes](#).

# 2

## Using the Outbound TCP Adapters

This page describes how to use the primary TCP outbound adapters (`EnLib.TCP.CountedOutboundAdapter`, `EnLib.TCP.CountedXMLOutboundAdapter`, and `EnLib.TCP.TextLineOutboundAdapter`).

### 2.1 Overview of Outbound TCP Adapter

InterSystems IRIS provides the following outbound TCP adapters, all of which are subclasses of `EnLib.TCP.OutboundAdapter`:

- `EnLib.TCP.CountedOutboundAdapter` contains methods to read or write blocks of data across the TCP connection. InterSystems IRIS acts as a TCP client exchanging blocks of data with a TCP listener. The block length specified in the first 4 bytes of the block. This convention allows a business operation to send requests to an external TCP server for fulfillment.
- `EnLib.TCP.CountedXMLOutboundAdapter` sends XML exported objects out as a counted block of bytes over a TCP connection and imports a response object.
- `EnLib.TCP.TextLineOutboundAdapter` contains methods to read or write text strings across a TCP connection. InterSystems IRIS acts as a TCP client exchanging blocks of data with a TCP listener. The default terminator for the text strings is the newline character (ASCII 10).

### 2.2 Overall Behavior

Within a production, an outbound adapter is associated with a business operation that you create and configure. The business operation receives a message from within the production, looks up the message type, and executes the appropriate method. This method usually executes methods of the associated adapter.

### 2.3 Creating a Business Operation to Use a TCP Outbound Adapter

To create a business operation to use a TCP outbound adapter, you create a new business operation class. Later, [add it to your production and configure it](#).

You must also create appropriate message classes, if none yet exist. See [Defining Messages](#).

The following list describes the basic requirements of the business operation class:

- Your business operation class should extend `Ens.BusinessOperation`.
- In your class, the *ADAPTER* parameter should equal `EnsLib.TCP.CountedOutboundAdapter`, `EnsLib.TCP.CountedXMLOutboundAdapter`, or `EnsLib.TCP.TextLineOutboundAdapter`.
- In your class, the *INVOCATION* parameter should specify the invocation style you want to use, which must be one of the following.
  - **Queue** means the message is created within one background job and placed on a queue, at which time the original job is released. Later, when the message is processed, a different background job is allocated for the task. This is the most common setting.
  - **InProc** means the message will be formulated, sent, and delivered in the same job in which it was created. The job will not be released to the sender's pool until the message is delivered to the target. This is only suitable for special cases.
- Your class must define a *message map* that includes at least one entry. A message map is an XData block entry that has the following structure:

```
XData MessageMap
{
<MapItems>
  <MapItem MessageType="messageclass">
    <Method>methodname</Method>
  </MapItem>
  . . .
</MapItems>
}
```

- Your class should define all the methods named in the message map. These methods are known as *message handlers*. Each message handler should have the following signature:

```
Method Sample(pReq As RequestClass, Output pResp As ResponseClass) As %Status
```

Here *Sample* is the name of the method, *RequestClass* is the name of a request message class, and *ResponseClass* is the name of a response message class. In general, the method code will refer to properties and methods of the Adapter property of your business operation.

For information on defining message classes, see [Defining Messages](#).

For information on defining the message handler methods, see [Creating Message Handler Methods](#).

- Your class can implement the **OnConnect()** callback method. This method is called each time the TCP outbound adapter establishes a new connection to or from a remote system.

If you implement **OnConnect()** method, it must have the following signature:

```
Method OnConnect(pTimeout As %Numeric) As %Status
```

Implement this method if you need to take some action each time a new connection is established, for example to send a logon sequence or a handshake token. The timeout argument is automatically provided by the TCP outbound adapter. It takes its value from the `ConnectTimeout` adapter setting. For details, see [Settings for the TCP Outbound Adapters](#).

If **OnConnect()** returns an error status, the new connection fails and the adapter is disconnected. If an untrapped exception occurs within **OnConnect()**, the adapter catches it and continues as if **OnConnect()** were not implemented.

- For other options and general information, see [Defining a Business Operation Class](#).

The following example shows the general structure that you need:

## Class Definition

```

Class ETCP.NewOperation1 Extends Ens.BusinessOperation
{
Parameter ADAPTER = "EnsLib.TCP.CountedOutboundAdapter";

Parameter INVOCATION = "Queue";

Method Sample(pReq As RequestClass, Output pResp As ResponseClass) As %Status
{
Quit $$$ERROR($$$NotImplemented)
}

XData MessageMap
{
<MapItems>
  <MapItem MessageType="RequestClass">
    <Method>Sample</Method>
  </MapItem>
</MapItems>
}
}

```

## 2.4 Creating Message Handler Methods

When you create a business operation class for use with a TCP outbound adapter, typically your biggest task is writing message handlers for use with this adapter, that is, methods that receive production messages and then write files.

Each message handler method should have the following signature:

```
Method Sample(pReq As RequestClass, Output pResp As ResponseClass) As %Status
```

Here *Sample* is the name of the method, *RequestClass* is the name of a request message class, and *ResponseClass* is the name of a response message class.

In general, the method should do the following:

1. Examine the inbound request message.
2. Using the information from the inbound request, call a method of the Adapter property of your business operation. For example, the following line invokes the **Connect()** method:

```
set sc=..Adapter.Connect()
```

See [Calling Adapter Methods from the Business Operation](#).

3. Make sure that you set the output argument (pOutput). Typically you set this equal to the response message. This step is required.
4. Return an appropriate status. This step is required.

### 2.4.1 Calling Adapter Methods from the Business Operation

#### Connect()

```
Method Connect(pTimeout As %Numeric) As %Status
```

Applies to all TCP outbound adapters.

**Connect()** opens a TCP socket to a port on the TCP listener machine. **Connect()** has one input argument, a %Numeric value that identifies the number of seconds to wait for the connection attempt to succeed. A typical call to the **Connect()** method gives this input argument the value of the configurable ConnectTimeout property.

If the Stay Connected adapter setting is set to -1, InterSystems IRIS creates the connection automatically at startup time, from within the **OnInit()** method. See [Reference for Settings](#).

### **Disconnect()**

```
Method Disconnect()
```

Applies to all TCP outbound adapters.

**Disconnect()** takes down the connection to the TCP listener.

### **SendMessageStream()**

```
Method SendMessageStream(pRequestStream As %Stream.Object,  
    ByRef pResponseStream As %CharacterStream = "%GlobalCharacterStream")  
    As %Status
```

Applies to `EnsLib.TCP.CountedOutboundAdapter`

**SendMessageStream()** sends the stream contents as a counted block on the TCP socket.

### **SendMessageString()**

```
Method SendMessageString(pRequestString As %String,  
    Output pResponseString As %String) As %Status
```

Applies to `EnsLib.TCP.CountedOutboundAdapter` and `EnsLib.TCP.TextLineOutboundAdapter`

**SendMessageString()** sends the string contents as a counted block on the TCP socket.

### **SendMessageXMLObj()**

```
Method SendMessageXMLObj(pRequest As %RegisteredObject,  
    Output pResponse As %RegisteredObject,  
    pResponseClassname As %String) As %Status
```

Applies to `EnsLib.TCP.CountedXMLOutboundAdapter`

**SendMessageXMLObj()** sends the request object to the TCP listener as a counted XML block. If the Get Reply adapter setting is True, **SendMessageXMLObj()** also gets a reply from the TCP listener as a counted XML block, which it imports as an InterSystems IRIS object. For details on Get Reply, see “[Reference for Settings](#).”

The `pResponseClassname` parameter is optional and requires the response object class definition to have a parameter called `RESPONSECLASSNAME` that equals a class name. The response class name is used to correlate the response data. It needs to be a persistent class that extends `%XML.Adaptor`.

### **TestConnection()**

```
Method TestConnection()
```

Applies to all TCP outbound adapters.

**TestConnection()** corrects the properties reflecting the connection state. If the adapter is connected, **TestConnection()** returns True. If not, **TestConnection()** calls **Disconnect()** and returns False.

## **2.5 Example for EnsLib.TCP.CountedOutboundAdapter**

The following is an example of a business operation class that references the `EnsLib.TCP.CountedOutboundAdapter`.

## Class Definition

```

Class Test.GeneralTCPOperation Extends Ens.BusinessOperation
{
  Parameter ADAPTER = "EnsLib.TCP.CountedOutboundAdapter";
  Parameter MAXREAD=30000;

  Method OnMessage(pReq As Test.GeneralSendRequest,
                  Output pResponse As Test.GeneralSendResponse) As %Status
  {
    Set str=
      pReq.Hr_pReq.DataLth_pReq.BID_pReq.Hr2_pReq.HrVacant_pReq.Cat_pReq.Hr3_pReq.Hr4
    Set tTextStream= ##class(%GlobalCharacterStream).%New()
    Set tSC=tTextStream.Write(str)
    Do pReq.Body2.Read(32) // Throw it away
    Set len=..#MAXREAD
    Set token= pReq.Body2.Read(.len)
    Set i=0
    While(len>0) {
      Set i=i+1
      Do tTextStream.Write(token)
      Set len=..#MAXREAD
      Set token= pReq.Body2.Read(.len)
    }
    Set tSC=
      ..Adapter.SendMessageStream(tTextStream,.tStreamReply) Quit:$$$ISERR(tSC) tSC
    Set pResponse=##class(Test.GeneralSendResponse).%New()
    Do tStreamReply.Rewind()
    Set pResponse.Body = tStreamReply.Read(,tSC)
    Quit tSC
  }
}

```

## 2.6 Adding and Configuring the Business Operation

To add your business operation to a production, use the Management Portal to do the following:

1. Add an instance of your business operation class to the production.
2. Configure the business operation. For information on the settings, see [Reference for Settings](#).
3. Enable the business operation.
4. Run the production.

## 2.7 Related Options

EnsLib.TCP contains additional TCP outbound adapters such as EnsLib.TCP.FramedOutboundAdapter. See the class reference for details.

InterSystems IRIS also provides specialized business service classes that use TCP adapters, and one of those might be suitable for your needs. If so, no programming would be needed. See [Connectivity Options](#).

You can develop a new inbound adapter class based on the EnsLib.TCP.OutboundAdapter or any of its subclasses. See [Creating Custom TCP Adapter Classes](#).



# 3

## Special Topics for TCP Adapters

This page discusses additional topics related to the TCP adapters.

For general information on customizing the callback methods of adapters, see [Less Common Tasks](#).

**Tip:** InterSystems IRIS® also provides specialized business service classes that use TCP adapters, and one of those might be suitable for your needs. If so, no programming would be needed. See [Connectivity Options](#).

### 3.1 Adding a TCP Status Service

The `EnsLib.TCP.StatusService` class parses an incoming text string to determine what type of production status information is being requested, and then produces a reply string suitable for writing out to the console screen. A user can interact with the status service directly, or write a command script that contacts the status service, issues commands, and writes the replies to a text file for later analysis.

A developer enables interactions via the status service as follows:

1. Add an instance of `EnsLib.TCP.StatusService` to the production.
2. Configure the instance of `EnsLib.TCP.StatusService` to accept communications via a particular TCP port. Set a Port number and a list of Allowed IP Addresses from which to accept connections. These are only two of the several settings that are available to configure the TCP text line inbound adapter associated with the status service. For details, see [Reference for Settings](#).

Once you have completed these steps, any time the status service is enabled and running, a user or command-line script can initiate a Telnet connection to the configured Port and send commands to the status service. Each command must be a text string that ends with the newline character (ASCII 10). The reply strings will also terminate with a newline.

The following table describes the supported command lines and the contents of the text string returned in each case.

Command Line	Text String Returned
build	The full name of the InterSystems IRIS software version.

Command Line	Text String Returned
<code>configitemstatus name</code>	When you enter this command with the configured <i>name</i> of any business host in the currently running production, the status service returns a string that expresses the current state of that business host.  A host that has currently running jobs, or active connections, records those activities in the returned string. If the identified host is not a member of the currently running production, the returned string indicates this.
<code>exit</code>	No string is returned. This command disconnects from the status service. You may enter <code>x</code> instead of <code>exit</code>
<code>localstarttime</code>	If InterSystems IRIS is running, this returns the start time of the currently running production, expressed in local time coordinates. Otherwise, it returns a string with the message <code>&lt;UNDEFINED&gt;</code>
<code>localtime</code>	The current time, in local time coordinates.
<code>namespace</code>	The interoperability-enabled namespace where the currently running production resides.
<code>production</code>	The configured name of the currently running production.
<code>quit</code>	No string is returned. This command disconnects from the status service. You may enter <code>q</code> instead of <code>quit</code>
<code>utcstarttime</code>	If InterSystems IRIS is running, this returns the start time of the currently running production, is expressed in universal time coordinates (UTC). Otherwise, it returns a string with the message <code>&lt;UNDEFINED&gt;</code>
<code>utctime</code>	The current time, in universal time coordinates (UTC).
<code>version</code>	The abbreviated name of the InterSystems IRIS software version, for example <code>2018.1.0.514/2171</code>
<code>x</code>	Same as <code>exit</code>

## 3.2 Creating Custom TCP Adapter Classes

To create a custom subclass of a TCP adapter class, do the following in an IDE:

1. On the **File** menu, click **New**, and then click the **General** tab.
2. Start the New Class Wizard by clicking **InterSystems IRIS Class Definition** and clicking **OK**.
  - a. Enter a package and class name and click **Next**.

**Important:** Be sure to avoid reserved package names; see [Reserved Package Names](#).
  - b. Click **Extends** for the **Class type**.
  - c. Click **Browse** next to **Name of super class** and navigate to the class that you want to subclass.
  - d. Click **OK**.
  - e. Click **Finish**.

The result is a template class like the following, depending on the class you chose:

### Class Definition

```
Class MyTest.NewClass1 Extends EnsLib.TCP.InboundAdapter
{
}
}
```

3. You can override any property, method, class parameter, or other class member inherited from the class or its parent classes, or you can add new class members. The only requirements are as follows:
  - An *inbound* adapter class must implement the **OnConnected()** method.  
In your implementation, use helper methods inherited by your class.
  - An *outbound* adapter class define methods that create a TCP connection, read or write over the connection, and disconnect.  
To define these methods, use helper methods inherited by your class.
4. Compile the class, which also saves it.

## 3.3 Common Customizations in TCP Adapter Subclasses

The following list suggests some common customizations in TCP adapter subclasses, in addition to implementing the methods that the adapter needs:

- You can define a different value for InitialExpressions for the Terminators property.  
To specify an ASCII character, use the ObjectScript function \$CHAR (abbreviated to \$C). The following example from EnsLib.TCP.TextLineInboundAdapter sets Terminators to the newline character (ASCII 10):

### Class Member

```
Property Terminators As %String [ InitialExpression = {$C(10)} ];
```

For more information about functions like \$CHAR, see ObjectScript Functions.

You can set Terminators to a single character or to a multi-character string. If you supply a string for the Terminators value, InterSystems IRIS uses the string as follows:

- Any one of the characters in the string serves to terminate the input.
- The complete string of characters is appended to the output.

- You can add properties and methods.
- You can add and remove settings. See [Adding and Removing Settings](#).
- If you need to require login credentials for the connection, simply add the property name `Credentials` to the `SETTINGS` list. The `Credentials` property is already defined as a `%String` in the base class `Ens.Adapter`.
- (For inbound adapters) You can also enforce a pool size of 1 by setting the parameter `SINGLEPOOLJOB = 1`:

```
/// Force a single listener job regardless of PoolSize setting
Parameter SINGLEPOOLJOB = 1;
```

Any subclass of this adapter class or a business service class that uses such an adapter class can use this parameter in its class code to better control the pool size.

- You can implement the **OnInit()** callback method to perform any special setup actions or initialize any structures.  
For inbound adapters, one of these actions might be to examine settings and initiate a connection between the adapter and the TCP client it will listen to, or to create any object properties that the adapter will use.  
For outbound adapters, one of these actions might be to examine settings and open a socket to a port on the TCP listener.
- For inbound adapters, you can implement the **OnConnected()** callback method.  
Whenever a connection exists to the configured TCP client, the adapter calls **OnConnected()** to read a stream from the TCP data source. This read operation is controlled by adapter settings.  
**OnConnected()** uses helper methods to parse the data.  
Upon successfully reading data, **OnConnected()** calls the production framework's **ProcessInput()** method to pass the received data stream to the associated business service. If this call returns an outbound stream, **OnConnected()** writes that data as a reply to the TCP client.  
If no data is available to read from the TCP connection during the CallInterval time, **OnConnected()** returns without doing anything.

# TCP Adapter Settings

This section provides reference information for the TCP adapters.

Also see [Settings in All Productions](#).

# Settings for the TCP Inbound Adapters

Provides reference information for settings of the TCP inbound adapters.

## Summary

The inbound TCP adapters have the following settings:

Group	Settings
Basic Settings	Port, Call Interval
Connection Settings	Job Per Connection, Allowed IP Addresses, OS Accept Connection Queue Size (QSize), Stay Connected, Read Timeout, SSL Configuration, Local Interface
Additional Settings	Accept Classnames, Charset

The remaining settings are common to all business services. For information, see [Settings for All Business Services](#).

## Accept Classnames

Applies to `EnsLib.TCP.CountedXMLInboundAdapter`.

Comma-separated list, giving the names of classes that this adapter will instantiate from XML blocks received.

## Allowed IP Addresses

Applies to all inbound TCP adapters.

A comma-separated list of remote IP addresses from which to accept connections. The adapter accepts IPV4 addresses, IPV6 addresses, or server names that the domain host controller can resolve. An optional *:port* designation is supported; so, for example, both of the following address formats are acceptable: `192.168.1.22` or `192.168.1.22:3298`.

**Note:** IP address filtering is a means to control access on private networks, rather than for publicly accessible systems. InterSystems does not recommend relying on IP address filtering as a sole security mechanism, as it is possible for attackers to spoof IP addresses.

If a port number is specified, connections from other ports will be refused.

If the string starts with an exclamation point (!) character, the inbound adapter initiates the connection rather than waiting for an incoming connection request. The inbound adapter initiates the connection to the specified address and then waits for a message. In this case, only one address may be given, and if a port is specified, it supersedes the value of the Port setting; otherwise, the Port setting is used.

## Call Interval

Applies to all inbound TCP adapters.

Number of seconds that the adapter will listen for incoming data from its configured source, before checking for a shutdown signal from the production framework.

If the adapter finds input, it acquires the data and passes it to the business service. The business service processes the data, and then the adapter immediately begins waiting for new input. This cycle continues whenever the production is running and the business service is enabled and scheduled to be active.

The default `CallInterval` is 5 seconds. The minimum is 0.1 seconds.

## Charset

Applies to `EnsLib.TCP.CountedInboundAdapter`, `EnsLib.TCP.CountedXMLInboundAdapter`, and `EnsLib.TCP.TextLineInboundAdapter`.

Specifies the character set of the inbound data. InterSystems IRIS® automatically translates the characters from this character encoding. The setting value is not case-sensitive. Use `Binary` for binary files, or for any data in which newline and line feed characters are distinct or must remain unchanged. Other settings may be useful when transferring text documents. Choices include:

- `Binary`—Binary transfer (the default)
- `Ascii`—Ascii mode FTP transfer but no character encoding translation
- `Default`—The default character encoding of the local InterSystems IRIS server
- `Latin1`—The ISO Latin1 8-bit encoding
- `ISO-8859-1`—The ISO Latin1 8-bit encoding
- `UTF-8`—The Unicode 8-bit encoding
- `UCS2`—The Unicode 16-bit encoding
- `UCS2-BE`—The Unicode 16-bit encoding (Big-Endian)
- Any other alias from an international character encoding standard for which NLS (National Language Support) is installed in InterSystems IRIS

For information on character sets and translation tables, see [Translation Tables](#).

## Endian

Applies to `EnsLib.TCP.CountedInboundAdapter` and `EnsLib.TCP.CountedXMLInboundAdapter`.

A choice of `Big` or `Little` indicates the byte order of the 4-byte block count prefix. `Big` endian means the most significant byte (MSB) goes over the wire first; `Little` endian means the least significant byte (LSB) goes over the wire first. The default value for this string is `Big`.

## Job Per Connection

Applies to all inbound TCP adapters.

When `True`, the adapter spawns a new job to handle each incoming TCP connection and allows simultaneous handling of multiple connections. When `False`, it does not spawn a new job for each connection. `False` is usually the appropriate choice for X12 connections. The default is `True`.

For TCP services, **JobPerConnection** causes each new incoming socket connection to be handled by a freshly spawned job rather than by the listener job itself. However, only one job at a time can be the listener, and one job must be the listener. So TCP services configured with **PoolSize** greater than 1 still only start one listener job. However, this listener can spawn an unlimited number of connection jobs if **JobPerConnection** is set to `True`.

If the **PoolSize** setting is configured to a value greater than 1, it serves as a limit on the number of simultaneous connection jobs that can exist. When this limit is reached, the listener does not accept any more connections until one or more of the existing connection jobs quits or dies. An Event Log warning appears when it first reaches the limit.

## Local Interface

Applies to all inbound TCP adapters.

Specifies the network interface through which the TCP connection should go. Select a value from the list or type a value. An empty value means use any interface.

## OS Accept Connection Queue Size (QSize)

Applies to all inbound TCP adapters.

Specifies the number of incoming connections the operating system should hold in reserve on behalf of this business service. Set to 0 if only 1 connection at a time is expected and subsequent connections ought to be refused immediately by the operating system. Set to a large number if you expect many clients to connect rapidly. The default is 100. The range is 0–1000, however the maximum incoming connections is dependent on the TCP implementations.

**Note:** If this is set to 0, a client attempting to connect will be refused connection if the business service is processing a previous incoming connection. This could lead to a scenario where a client is connected to the business service, but disconnects and attempts to reconnect again in the brief period before the listening socket is reopened. As a result, the client will not try to reconnect.

For HL7 and X12 TCP inbound adapters, the default is set to 0. This supports the HL7 First In, First Out (FIFO) strategy where only one connection at a time is expected.

## Port

Applies to all inbound TCP adapters.

Identifies the TCP port on the local machine where the adapter is listening for TCP requests. Avoid specifying a port number that is in the range used by the operating system for ephemeral outbound connections.

## Read Timeout

Applies to all TCP adapters.

Number of seconds to wait for each successive incoming TCP read operation, following receipt of initial data from the remote TCP port. The default is 5 seconds. The range is 0–600 seconds (a maximum of 10 minutes).

## SSLConfig

Applies to all TCP adapters.

The name of an existing TLS configuration to use to authenticate this connection. Choose a client TLS configuration, because the adapter initiates the communication.

To create and manage TLS configurations, use the Management Portal. See *InterSystems TLS Guide*. The first field on the **Edit SSL/TLS Configuration** page is **Configuration Name**. Use this string as the value for the **SSLConfig** setting.

## Stay Connected

Applies to all TCP adapters.

Specifies how long the adapter stays connected to a remote system after an input event and determines how the adapter handles disconnections.

You can specify any of the following values:

- A positive value—The adapter remains connected to the remote system for this number of seconds. Disconnections are not handled like errors.
- 0—The adapter disconnects immediately. Disconnections are not handled like errors.
- –1—The adapter is permanently connected, including during idle times. Disconnections are handled like errors.

**Note:** Adapters auto-connect at startup only if the **Stay Connected** value is –1.

The default value is –1.

**Important:** If you set **Stay Connected** to **-1** and **Job Per Connection** to **False**, then the adapter may not detect disconnections caused by network occurrences. Specifically, the following situation may arise:

1. The adapter accepts an incoming connection from a remote system.

Recall that when **Job Per Connection** is set to **False**, the listener job on the adapter handles each incoming connection rather than spawning new jobs to do so.

2. A network occurrence causes a disconnection, and the adapter does not receive a TCP FIN or RST packet to terminate the connection.

For example, a firewall between the remote system and adapter may close the connection after a certain amount of time.

3. The remote system attempts to reestablish the connection, but cannot connect to the listening socket on the adapter.
4. The remote system generates errors and continuously attempts to reestablish the connection.

To remedy the situation, you must stop and then restart the business service that uses the adapter.

To prevent the situation, set **Stay Connected** to a value other than **-1**. For example, if you set **Stay Connected** to **300**, the adapter disconnects after five minutes. Additionally, if you do not need the adapter to process requests in order, you can set **Job Per Connection** to **True**, enabling the adapter to process multiple incoming connection requests at one time.

# Settings for the TCP Outbound Adapters

Provides reference information for settings of the TCP outbound adapters.

## Summary

The outbound TCP adapters have the following settings:

Group	Settings
Basic Settings	<a href="#">IP Address</a> , <a href="#">Port</a>
Connection Settings	<a href="#">Stay Connected</a> , <a href="#">Connect Timeout</a> , <a href="#">Reconnect Retry</a> , <a href="#">Get Reply</a> , <a href="#">Response Timeout</a> , <a href="#">Read Timeout</a> , <a href="#">SSL Configuration</a> , <a href="#">Local Interface</a> , <a href="#">Charset</a> , <a href="#">Endian</a>
Additional Settings	<a href="#">Charset</a>

The remaining settings are common to all business operations. For information, see [Settings for All Business Operations](#).

## Charset

Applies to `EnsLib.TCP.CountedOutboundAdapter`, `EnsLib.TCP.CountedXMLOutboundAdapter`, and `EnsLib.TCP.TextLineOutboundAdapter`.

Specifies the desired character set of the outbound data. InterSystems IRIS® automatically translates the characters to this character encoding. See [Charset](#).

## Connect Timeout

Applies to all outbound TCP adapters.

Number of seconds to wait on each attempt to connect to the remote TCP listener. The default is 5 seconds.

## Endian

Applies to `EnsLib.TCP.CountedOutboundAdapter` and `EnsLib.TCP.CountedXMLOutboundAdapter`.

A choice of `Big` or `Little` indicates the byte order of the 4-byte block count prefix. `Big` endian means the most significant byte (MSB) goes over the wire first; `Little` endian means the least significant byte (LSB) goes over the wire first. The default value for this string is `Big`.

## Get Reply

Applies to all outbound TCP adapters.

If 1 (true) wait to read a reply message back from the socket before returning. If 0 (false) do not wait. The default value is 1.

## IP Address

Applies to all outbound TCP adapters.

IP address to make a TCP connection to. The adapter accepts an IPV4 address, an IPV6 address, or a server name that the domain host controller can resolve.

If the string starts with an exclamation point (!) character, the outbound adapter does not initiate the connection but rather waits for an incoming connection request. Once it accepts the connection request, it can send outbound messages. The exclamation point can be specified by itself or followed by a comma-separated list of IP addresses. If the exclamation point is not followed by an IP address, the outbound TCP adapter accepts a connection from any IP address, otherwise the adapter

accepts connections only from the specified addresses. The adapter accepts IPV4 addresses, IPV6 addresses, or server names that the domain host controller can resolve. You can optionally specify a port for each IP address by appending a colon (:) followed by the port number. If a port is specified, the adapter accepts connections from only the specified port.

**Note:** IP address filtering is a means to control access on private networks, rather than for publicly accessible systems. InterSystems does not recommend relying on IP address filtering as a sole security mechanism, as it is possible for attackers to spoof IP addresses.

## LocalInterface

Applies to all outbound TCP adapters.

Specifies the network interface through which the TCP connection should go. Select a value from the list or type a value. An empty value means use any interface. For details, see [Local Interface](#).

## Port

Applies to all outbound TCP adapters.

Identifies the TCP port to connect to.

## QSize

Applies to `EnsLib.TCP.CountedOutboundAdapter`, `EnsLib.TCP.CountedXMLOutboundAdapter`, and `EnsLib.TCP.TextLineOutboundAdapter`.

Specifies the number of incoming connections the operating system should hold in reserve on behalf of this business operation. There is generally no reason to change the default value of 0. Either the connection is outbound, or the business operation is using '!' inbound mode, in which case only 1 client is allowed at a time, per business operation pool job.

## Read Timeout

Applies to all TCP adapters.

Number of seconds to wait for each successive incoming TCP read operation, following receipt of initial data from the remote TCP port. The default is 5 seconds. The range is 0–600 seconds (a maximum of 10 minutes).

## Reconnect Retry

Applies to all outbound TCP adapters.

Number of times the adapter attempts to send a message before dropping the connection and reconnecting. A value of 0 (zero) means that the adapter attempts to send the message forever without disconnecting. The default value is 5.

## Response Timeout

Applies to all outbound TCP adapters.

Number of seconds to wait for a response to begin arriving back from the remote system after sending a request. The default is 15 seconds.

## SSL Configuration

Applies to all TCP adapters.

The name of an existing TLS configuration to use to authenticate this connection. Choose a client TLS configuration, because the adapter initiates the communication. For details, see [SSL Config](#).

## Stay Connected

Applies to all TCP adapters.

Specifies how long the TCP adapter stays connected to the remote system after completing an operation and determines how the adapter handles disconnections.

You can specify any of the following values:

- A positive value—The adapter remains connected to the remote system for this number of seconds. Disconnections are not handled like errors.
- 0—The adapter disconnects immediately. Disconnections are not handled like errors.
- -1—The adapter is permanently connected, including during idle times.

**Note:** Adapters auto-connect at startup only if the **Stay Connected** value is -1. Disconnections are handled like errors.

The default value is -1.