

SocketServerXtra Version 1

www.xtramania.com

© Eugene Shoustrov, 2006

All trademarked names mentioned in this document and product are used for editorial purposes only, with no intention of infringing upon the trademarks.

SocketServerXtra	1
About SocketServerXtra	1
About NetCompanionSet	1
'NetCompanionSet' Xtras License Agreement	1
SocketServerXtra Programmer's Guide.....	3
How to Use SocketServerXtra	3
Server side	3
Client side (synchronous)	4
Client side (asynchronous)	5
Debugging and Errors Handling.....	5
Lingo Errors.....	6
Programming Errors	6
Simple Debugging Mode.....	6
Advanced Debugging Mode.....	6
Using Put Command.....	7
Using Debugger and Object Inspector	7
SocketServerXtra Programmer's Reference	8
Main Objects	8
Common Features of SocketServerXtra Objects	9
Error Handling Support	9
<i>Succeeded</i>	9
<i>Failed</i>	10
<i>LastErrorCode</i>	10
<i>LastError</i>	11
Debugging Support.....	11
<i>DebugMode</i>	12
Server Object	14
Methods	14
<i>Interface()</i>	14
<i>Start(nPortNumber)</i>	15
<i>ReadNext()</i>	15
<i>Reply(valData)</i>	16
<i>DropConnection(strAddress)</i>	16
Properties	17
<i>PendingMessagesCount</i>	17
<i>Port</i>	17
<i>Connections</i>	17
<i>Data</i>	17
<i>RemoteTierAddress</i>	18
<i>Subject</i>	18
<i>SenderId</i>	18
Synchronized Client Object.....	19
Methods	19

<i>Interface()</i>	19
<i>Connect(strServer, nPortNumber, strSenderId)</i>	19
<i>Send(strSubject, valData)</i>	20
<i>Calling other methods</i>	21
Properties	21
<i>Server</i>	21
<i>Port</i>	21
Xtra-level methods	22
<i>Init(nDebug)</i>	22
<i>New(symObjectType)</i>	22
<i>Version()</i>	24
Data types supported	24

SocketServerXtra

About SocketServerXtra

SocketServerXtra extends the Macromedia Director's Lingo functionality with networking capabilities. The xtra uses Shockwave Multi-user Server protocol for network messages and may become a simple MUS replacement. SocketServerXtra allows making normal Director movie into Internet server that handles remote clients with normal Lingo.

SocketServerXtra is available for Macromedia Director (v7 and later) under Windows 95/98/ME/NT/2000/XP.

SocketServerXtra is available for Shockwave.

Note: All trademarked names mentioned in this document and product are used for editorial purposes only, with no intention of infringing upon the trademarks.

About NetCompanionSet

SocketServerXtra is shipped within NetCompanionSet. It is a bundle of xtras that provide networking support for Macromedia Director. The set currently consists of SocketServerXtra only.

'NetCompanionSet' Xtras License Agreement

This user license agreement (the AGREEMENT) is an agreement between you (individual or single entity) and MediaMacros, Inc. and Eugene Shoustrov for the included 'NetCompanionSet' XTRAS (the SOFTWARE) that are accompanying this AGREEMENT.

The SOFTWARE is the property of Eugene Shoustrov and is protected by copyright laws and international copyright treaties. The SOFTWARE is not sold, it is licensed.

LICENSED VERSION

The LICENSED VERSION means a Registered Version (using your personal registration number) or an original fully working version of the SOFTWARE. If you accept the terms and conditions of this AGREEMENT, you have certain rights and obligations as follow:

YOU MAY:

1. Install and use the SOFTWARE on any single computer.
2. Make a copy of the SOFTWARE for archival purposes only.
3. Distribute an unlimited number of copies of the Xtra with your final runtimes provided that the source code is protected and your serial number is not accessible to any 3rd party.

YOU MAY NOT:

1. Copy and distribute the SOFTWARE with an accessible serial number.
2. Sublicense, rent or lease your license

3. Decompile, disassemble, reverse engineer or modify the SOFTWARE or any portion of it, or make any attempt to bypass, unlock, or disable any protective or initialization system on the SOFTWARE.
4. Copy the documentation accompanying the SOFTWARE for use in other software.

WARRANTY DISCLAIMER

The SOFTWARE is supplied "AS IS". MediaMacros, Inc. and Eugene Shoustrov disclaim all warranties, expressed or implied, including, without limitation, the warranties of merchantability and of fitness for any purpose. The user must assume the entire risk of using this SOFTWARE.

DISCLAIMER OF DAMAGES

MediaMacros, Inc. and Eugene Shoustrov assume no liability for damages, direct or consequential, which may result from the use of this SOFTWARE, even if MediaMacros, Inc. and/or Eugene Shoustrov have been advised of the possibility of such damages.

TERM

This license is effective from the date of obtaining or purchasing the SOFTWARE and shall remain in force until terminated. You may terminate the license and this agreement at any time by destroying the SOFTWARE and its documentation, together with all copies in any form that reside on your computer or media.

COPYRIGHT NOTICE:

The Company and/or our Licensors hold valid copyright in the Software. Nothing in this Agreement constitutes a waiver of any rights under U.S. Copyright law or any other federal or state law.

ACKNOWLEDGMENT:

BY USING THIS SOFTWARE YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU ALSO AGREE THAT THIS AGREEMENT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN YOU AND THE COMPANY AND SUPERCEDES ALL PROPOSALS OR PRIOR ENDORSEMENTS, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN YOU AND THE COMPANY OR ANY REPRESENTATIVE OF THE COMPANY RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

All trademarked names mentioned in this document and product are used for editorial purposes only, with no intention of infringing upon the trademarks.

SocketServerXtra Programmer's Guide

How to Use SocketServerXtra

SocketServerXtra extends communication capabilities of normal Director Projectors. Actually it allows turning usual Projector into network server application that can handle multiple network clients.

SocketServerXtra uses Shockwave MUS (Multi User Server) protocol for network messages, so it can work with multiuser xtra shipped with Director 8.5 and all modern versions of Shockwave player. It means that you can connect to SocketServerXtra's server from any Shockwave movie or Projector on both Mac and Windows platforms.

SocketServerXtra also provides its own client interface that can be used instead of multiuser xtra.

Server side

[Server object](#) is a key component of SocketServerXtra. Once created and started it serves as Internet server by listening connection requests to the specified port, receiving and sending data according to MUS protocol.

To create server object use xtra-level [New](#) method:

```
objServer = xtra("SocketServerXtra").new( #Server )
```

Use [Start\(nPortNumber \)](#) method to start listening the incoming connections and messages on the specified port. Calling this method makes server object to open listening socket on the specified port and create the thread for handling network messages asynchronously. While Director's main thread is running your Lingo code, server's thread may receive incoming messages and send outgoing messages.

The SocketServerXtra automatically handles incoming connection requests and places incoming messages into the queue. Your Lingo code should handle incoming messages by picking them up from the queue one by one and replying to them. Once server is running you should periodically retrieve incoming messages from the message queue and reply to them. Use [ReadNext\(\)](#) method to retrieve the next message from the queue. It returns true if the next message is available.

Use [Subject](#) and [SenderId](#) properties to get these properties of the current message. [Data](#) property returns the Lingo value passed as a contents of the message.

Use [Reply\(valData \)](#) method to reply to the current message. Otherwise you may call [DropConnection\(\)](#) to drop connection with the client who posted the current message.

[Connections](#) property returns the list of addresses of currently connected clients. Using [DropConnection\(strAddress \)](#) you may drop the connection with specific client.

This sample shows basic server functionality:

```
property m_objServer
on beginSprite me
    net = xtra("SocketServerXtra")
```

```
m_objServer = net.new( #Server )
-- Start server bound to the port 4545
m_objServer.Start( 4545 )
end
on exitFrame me
-- Read available incoming messages one by one
repeat while objServer.ReadNext()
-- Read what is passed
valIncomingData = oServer.Data
strSubject = oServer.Subject
-- Prepare reply based on incoming data
valOutcomingData = valIncomingData
-- Reply to the message
oServer.Reply( valOutcomingData )
end repeat
end
```

Client side (synchronous)

[Synchronized client object](#) is the simple client side for the communication with the Server object over Internet. Similar to multiuser xtra, it can send messages to the server and receive what it replies. 'Synchronized' means that by sending data to the server client is blocked until server replies or connection is dropped.

To create synchronized client object use xtra-level [New](#) method:

```
objClient = xtra("SocketServerXtra").New( #SyncClient )
```

Use [Connect\(strServer, nPortNumber, strSenderId \)](#) method to establish connection to the specified server.

Once connection is established you may communicate with the server via [Send\(valData \)](#) method. Synchronized client is intended for use in remote method call scenario, thus `Send(valData)` returns only when the client has received the server's response. This scenario goes well for local area network (LAN) applications, although it may be used in wide area network (WAN) applications too in some cases.

The code below shows how to ask server for something:

```
net = xtra("SocketServerXtra")
objConnection = net.new( #SyncClient )
-- Connect to the server
objConnection.Connect( "server_name_or_ip", 4545, "UserName" )
-- Prepare data to be sent
valSomeLingoValue = [ 1, 2, "3", [1, 2], [#propName:propValue], #etc ]
-- Sending data and waiting while server replies
valServerReply = objConnection.SomeSubject( valSomeLingoValue )
-- Seeing what server has replied
put valServerReply
```

Note that synchronized client object automatically maps unknown method names to [Send](#) method with appropriate subject. So calling: `cnn.SomeMethod(valData)` is the same as:

```
cnn.Send( "SomeMethod", valData )
```

This mapping may simplify programming network applications by simulating remote procedure calling. Client side calls normal Lingo method and receives normal Lingo return value, while internally network message is generated and passed to the server and back.

Client side (asynchronous)

You may use native multiuser xtra shipped with Shockwave player or Director 8.5. Shockwave xtras are normally installed here:

```
C:\WINDOWS\system32\Macromed\Shockwave 10\Xtras
```

Using multiuser xtra requires parent script object (with name "NetHandler") that will handle incoming messages. It may look like this:

```
global oConnection
-- StandartHandler
on defaultCallback me
    msg = oConnection.getNetMessage()
    if msg[#ErrorCode] then
        put oConnection.GetNetMessageString( msg[#ErrorCode] )
    else
        put string(msg) & RETURN
    end if
end
```

To establish connection with the server use:

```
net = xtra("multiuser")
-- Create new client instance
mus = net.new()
-- Prepares messages handler
mus.setNetMessageHandler( #defaultCallback, script("NetHandler") )
-- Connects to the server
mus.connectToNetServer( "SenderId", "", "host", 4545, "MovieName" )
```

On `connectToNetServer` method multiuser xtra sends a message with subject ["Logon"](#). Server side Lingo should reply to this message to make multiuser xtra think it is connected successfully. After this reply is received you may send usual messages to server and handle messages posted back to client.

```
-- Prepare data to be sent
valSomeLingoValue = [ 1, 2, "3", [1, 2], [#propName:propValue], #etc ]
mus.sendNetMessage( "Dummy", "Subject", valSomeLingoValue )
```

Reply will be handled by "NetHandler" parent script object.

Debugging and Errors Handling

There are two main levels of errors related to SocketServerXtra. They have completely different nature and therefore have to be handled differently.

Lingo Errors

Lingo errors are similar to incorrect Lingo syntax run-time errors. They cause Director to show error alert saying something like "Method or property not found in object" or "One parameter expected". In Projector they might halt script execution etc. These errors usually mean that something is wrong with the programming. Wrong method call syntax is used or something similar to it. SocketServerXtra might return error codes to Director that make Director to show Lingo error alert box. It happens when object discovers the programming error at the Lingo level (wrong syntax, wrong parameters and other compile time evident programming errors).

Programming Errors

This level includes errors that are actually exception conditions. They happen or do not happen depending on particular execution context. They are normal in programming practice and have to be handled programmatically. For example if file operation fails it does not have to worry end-user with Lingo error alert box. Instead developer should check whether operation completed successfully and perform what is appropriate.

SocketServerXtra provides programming errors handling support based on storing status of the last call within every object. In other words, every SocketServerXtra's object keeps the error code and description returned by the most recently called method or property. Before returning from the call to any object the last error information (if any) is being set by the object. Right before calling the next method or property of the object the last error information is cleared.

To check the status of the most recent call to the object use [obj.Failed](#) or [obj.Succeeded](#) properties. The error message and error code are available via [obj.LastError](#) and [obj.LastErrorCode](#) properties.

Simple Debugging Mode

Since errors are happening SocketServerXtra provides debugging modes to simplify debugging process.

In simple debugging mode any object puts error information into Messages window whenever error occurred. Usually simple debugging mode is useful to detect whether script is executed well or there is a problem somewhere. Error messages usually come from wrapped objects but there is no information about the context where error occurred.

To set the simple debugging mode for the xtra use:

```
on prepareMovie
  if the playerMode = "author" then
    xtra("SocketServerXtra").Init( 1 )
  end if
end
```

Advanced Debugging Mode

Advanced debugging mode allows you to catch error right in Debugger whenever error occurred. In this mode SocketServerXtra tries to call movie-level handler `SocketServerXtra_DebugEvent(strMes, nCode)`. If there is no such handler, the xtra behaves as in simple debugging mode. This handler may contain any Lingo statements. Furthermore, you can place a break point inside this handler and use Director's debugging capabilities to view the calling context, variables etc.

Sample movie-level handler for advanced debugging.

```
on prepareMovie
  if the playerMode = "author" then
    xtra("SocketServerXtra").Init( 2 )
  end if
end

on SocketServerXtra_DebugEvent strMes, nCode
  put strMes -- Place the break point here
end
```

Debugging mode is kept separately for every SocketServerXtra object. Use [DebugMode](#) property to change the debugging mode of the particular object directly. Otherwise use xtra-level [Init\(nDebug \)](#) method to set the default debugging mode for newly created objects. This method does not affect objects that already exist at the time of calling this method.

Using Put Command

Every object provides descriptive information about itself via put Lingo method. To see what the object contains simply put it in Messages window.

```
net = xtra("SocketServerXtra")
objServer = net.new( #Server )

put objServer
--"< SocketServerXtra, SocketServer >"

objServer.Start( 4545 )

put objServer
--"< SocketServerXtra, SocketServer, Port: 4545 >"
```

Using Debugger and Object Inspector

SocketServerXtra objects support viewing their contents via Director Debugger and Object Inspector.

Automation allows expanding its entry in Debugger to view properties of the wrapped Automation object. It is quite convenient although it has side effect that conflicts with debugging modes. When object's entry in debugger is expanded Director internally calls all properties available to view in debugger. Xtra cannot distinguish whether it is called by debugger or by Lingo script. Therefore last error information kept by the object is erased with the status of the last method or property that was called by Director but not Lingo script. In advanced debugging mode the SocketServerXtra_DebugEvent movie level handler could be called while Director asks object for its property values. So take care with that.

SocketServerXtra Programmer's Reference

Main Objects

SocketServerXtra provides several types of Lingo objects.

[Server object](#) is a key component of SocketServerXtra. Once created and started it serves as Internet server by listening connection requests to the specified port, receiving and sending data according to MUS protocol.

To create server object use xtra-level [New](#) method:

```
objServer = xtra("SocketServerXtra").new( #Server )
```

[Synchronized client object](#) is the simple client side for the communication with the Server object over Internet. Similar to multiuser xtra, it can send messages to the server and receive what it replies. Since it uses MUS protocol, it is possible to use multiuser xtra to connect to the server part of SocketServerXtra.

To create synchronized client object use xtra-level [New](#) method:

```
objClient = xtra("SocketServerXtra").New( #SyncClient )
```

Further versions of SocketServerXtra might include other object types as well.

Common Features of SocketServerXtra Objects

Every SocketServerXtra's object is built on the same prototype that provides some basic functionality common for all such objects implemented by xtra. Basic functionality includes [error handling](#) and [debugging](#) support.

Error Handling Support

There are two main levels of errors related to SocketServerXtra. They have completely different nature and therefore have to be handled differently.

Lingo Errors

Lingo errors are similar to incorrect Lingo syntax run-time errors. They cause Director to show error alert saying something like "Method or property not found in object" or "One parameter expected". In Projector they might halt script execution etc. These errors usually mean that something is wrong with the programming. Wrong method call syntax is used or something similar to it. SocketServerXtra might return error codes to Director that make Director to show Lingo error alert box. It happens when object discovers the programming error at the Lingo level (wrong syntax, wrong parameters and other evident programming errors).

Programming Errors

This level includes errors that are actually exception conditions. They happen or do not happen depending on particular execution context. They are normal in programming practice and have to be handled programmatically. For example if file operation fails it does not have to worry end-user with Lingo error alert box. Instead developer should check whether operation completed successfully and perform what is appropriate.

SocketServerXtra provides programming errors handling support based on storing status of the last call within every object. In other words, every SocketServerXtra's object keeps the error code and description returned by the most recently called method or property. Before returning from the call to any object the last error information (if any) is being set by the object. Right before calling the next method or property of the object the last error information is cleared.

To check the status of the most recent call to the object use [obj.Failed](#) or [obj.Succeeded](#) properties. The error message and error code are available via [obj.LastError](#) and [obj.LastErrorCode](#) properties.

Succeeded

Returns whether the most recent call to the wrapped contents was successful.

Syntax

```
bResult = obj.Succeeded
```

Return values

True

If the previous call to the object was successful

False

If the previous call to the object was not successful. The error code and description are available via [LastErrorCode](#) and [LastError](#) properties.

Remarks

This property as well as other properties described in this section does not clear the last error flag. It means this property does not affect the last error information for the particular object.

Failed

Returns whether the most recent call to the wrapped contents has failed.

Syntax

```
bResult = obj.Failed
```

Return values

True

If the previous call to the object was not successful. The error code and description are available via `LastErrorCode` and `LastError` properties.

False

If the previous call to the object was successful

Remarks

This property as well as other properties described in this section does not clear the last error flag. It means this property does not affect the last error information for the particular object.

LastErrorCode

Returns the code of the last error (if any) happened while calling the object.

Syntax

```
nCode = obj.LastErrorCode
```

Return values

Integer

Integer value that indicates the error code of the most recent call to the wrapped contents. If the most recent call completed successfully, the error code is 0.

Remarks

This property as well as other properties described in this section does not clear the last error flag. It means this property does not affect the last error information for the particular object.

Most of error codes are coming from the wrapped Automation objects. They define their own error codes usually described in component's documentation.

Other error codes are defined by COM. Here come errors produced by passing incorrect parameters or skipping required parameter etc.

Several error codes are defined by SocketServerXtra. They could occur if SocketServerXtra failed to typecast Lingo value into COM variant or vice versa.

LastError

Returns the description of the last error (if any) happened while calling the contents of a object.

Syntax

```
strErrorMessage = obj.LastError
```

Return values

String

String value that contains the error description of the most recent call to the wrapped contents. If the most recent call completed successfully, the error description is empty.

Remarks

This property as well as other properties described in this section does not clear the last error flag. It means this property does not affect the last error information for the particular object.

Debugging Support

Every object created by SocketServerXtra can detect errors returned by wrapped objects. Internal SocketServerXtra errors (type casting problems etc) could happen too. Normally these errors could be trapped programmatically by checking object's last error status after any meaningful call to the object. See [error handling](#) support properties for more details. To simplify debugging process SocketServerXtra provides debugging mode.

Simple Debugging Mode

In simple debugging mode any object puts error information into Messages window whenever error occurred. Usually simple debugging mode is useful to detect whether script is executed well or there is a problem somewhere. Error messages usually come from wrapped objects but there is no information about the context where error occurred.

Advanced Debugging Mode

Advanced debugging mode allows you to catch error right in Debugger whenever error occurred. In this mode SocketServerXtra tries to call movie-level handler `SocketServerXtra_DebugEvent(strMes, nCode)`. If there is no such handler, the xtra behaves as in simple debugging mode. This handler may contain any Lingo statements. Furthermore, you can place a break point inside this handler and use Director's debugging capabilities to view the calling context, variables etc.

Sample movie-level handler for advanced debugging.

```
on prepareMovie
  if the playerMode = "author" then
    xtra("SocketServerXtra").Init( 2 )
  end if
end

on SocketServerXtra_DebugEvent strMes, nCode
  put strMes -- Place the break point here
end
```

Debugging mode is kept separately for every SocketServerXtra object. Use [DebugMode](#) property to change the debugging mode of the particular object directly. Otherwise use xtra-level [Init\(nDebug \)](#) method to set the default debugging mode for newly created objects. This method does not affect objects that already exist at the time of calling this method.

Using Put Command

Every object provides descriptive information about itself via put Lingo method. To see what the object contains simply put it in Messages window.

```
net = xtra("SocketServerXtra")
objServer = net.new( #Server )

put objServer
--"< SocketServerXtra, SocketServer >"

objServer.Start( 4545 )

put objServer
--"< SocketServerXtra, SocketServer, Port: 4545 >"
```

Using Debugger and Object Inspector

SocketServerXtra objects support viewing their contents via Director Debugger and Object Inspector.

SocketServerXtra object allow expanding its entry in Debugger to view its properties. It is quite convenient although it has side effect that conflicts with debugging modes. When object's entry in debugger is expanded Director internally calls all properties available to view in debugger. Object cannot distinguish whether it is called by debugger or by Lingo script. Therefore last error information kept by the object is erased with the status of the last method or property that was called by Director but not Lingo script. In advanced debugging mode the `SocketServerXtra_DebugEvent` movie level handler could be called while Director asks object for its property values. So take care with that.

DebugMode

Sets or gets the debugging mode for the specific object.

Syntax

```
nDebugMode = obj.DebugMode
obj.DebugMode = nDebugMode
```

Parameters

nDebugMode - Integer

Debugging mode for newly created objects. This parameter can be one of the following values.

Value	Meaning
0	No debugging support. Release behavior.
1	Simple debugging. Any error is automatically printed in Messages window.
2	Advanced debugging. When any error is occurred, the xtra calls movie level handler <code>SocketServerXtra_DebugEvent(strMes, nCode</code>

Value	Meaning
-------	---------

).

Return values

Integer

Integer value that indicates the current debugging mode applied to the object.

Remarks

This property as well as other properties described in this section does not clear the last error flag. It means this property does not affect the last error information for the particular object.

Debugging mode is inherited by objects that are produced by the current object during calls to the wrapped contents.

Temporary objects produced by cascading properties access Lingo statement get the debugging mode from their parent object.

New objects created by [New](#) xtra-level method inherits default xtra-level debug mode that is set by [Init](#) method.

Server Object

Server object is a key component of SocketServerXtra. Once created and started it serves as Internet server by listening connection requests to the specified port, receiving and sending data according to MUS protocol.

To create server object use xtra-level [New](#) method:

```
objServer = xtra("SocketServerXtra").new( #Server )
```

Use [Interface\(\)](#) method to get the short description of methods and properties provided by the object.

Use [Start\(nPortNumber \)](#) method to start listening the incoming connections and messages on the specified port.

Once server is running you should periodically retrieve incoming messages from the message queue and reply to them. Use [ReadNext\(\)](#) method to retrieve the next message from the queue. It returns `true` if the next message is available and become the current one.

Several methods and properties are applied to the current message being handled. Use [Subject](#) and [SenderId](#) properties to get these properties of the current message. [Data](#) property returns the Lingo value passed as a contents of the message. [RemoteTierAddress](#) property returns the address string of the remote tier whose message is being processed.

Use [Reply\(valData \)](#) method to reply to the current message. Otherwise you may call [DropConnection\(\)](#) to drop connection with the client who posted the current message.

[Connections](#) property returns the list of addresses of currently connected clients. Using [DropConnection\(strAddress \)](#) you may drop the connection with specific client.

Methods

Interface()

Returns the short description of methods and properties provided by the object.

Syntax

```
strDescription = objServer.Interface()
```

Return values

String

Returns a string with short listing of methods and properties exposed by this object.

Remarks

This method does not clear the last error flag. It means this method does not affect the last error information for the particular object.

Sample

The sample shows how to get create a new object instance and view its interface description.

```
net = xtra( "SocketServerXtra" )
```

```
objServer = net.New( #Server )  
put objServer.Interface()
```

Start(nPortNumber)

Starts the server on the specified port.

Syntax

```
objServer.Start( nPortNumber )
```

Parameters

nPortNumber

Integer port number used to open listening socket available for incoming connections.

Remarks

Once this method is called, server object opens listening socket on the specified port and creates the thread for handling network messages asynchronously. While Director's main thread is running your Lingo code, server's thread may receive incoming messages and send outgoing messages.

The SocketServerXtra automatically handles incoming connection requests and places incoming messages into the queue. Your Lingo code should handle incoming messages by picking them up from the queue one by one and replying to them.

Sample

This sample shows basic server functionality:

```
on exitFrame me  
  -- Read available incoming messages one by one  
  repeat while objServer.ReadNext()  
    -- Read what is passed  
    valIncomingData = oServer.Data  
    strSubject = oServer.Subject  
    -- Prepare reply based on incoming data  
    valOutcomingData = valIncomingData  
    -- Reply to the message  
    oServer.Reply( valOutcomingData )  
  end repeat  
end
```

ReadNext ()

Reads the next incoming message if any. Normally server movie should call this method periodically to process incoming messages.

Syntax

```
bSuccessful = objServer.ReadNext()
```

Returns

bSuccessful

Boolean value that indicates whether the next message is ready for handling. Normally you may call this method in a loop while it returns true. When it returns false, then there are no messages for processing at the moment.

Remarks

If this method returns true you should handle the current message. Note that synchronized client relies on server handling messages as quick as possible, since synchronized client blocks the movie until server replies something. So, server should respond somehow on every valid incoming message. Server side may either reply to the message with [Reply\(valData \)](#) or drop the client connection with [DropConnection\(\)](#) method. If the server does not reply to client message, synchronized client will hang until server drops its connection.

Asynchronous client (for example [using multiuser xtra](#)) are based on asynchronous messages, so they may not require server to respond. In this case it is up to you whether to reply or not.

Reply(valData)

Replies to the current message with the specified data.

Syntax

```
objServer.Reply( valData )
```

Parameters

valData

Lingo value to be passed back to client. Synchronized client will get this data as a return value of its [Send\(valData \)](#) method call. Asynchronous client will get this data as normal incoming message from server. See [supported data types](#) for details of what type can be this Lingo value.

DropConnection(strAddress)

Drops the connection to the specified client.

Syntax

```
objServer.DropConnection( Optional strAddress )
```

Parameters

strAddress

String identifying the client's address. If this argument is omitted, then connection to the current message client is dropped.

Remarks

Normally client address string consist of its IP address and port: "IP:port". Existing client addresses are available via [Connections](#) property. Also while processing message from a client, you can obtain its address using [RemoteTierAddress](#) property.

Properties

PendingMessagesCount

Returns the number of messages in the queue waiting to be handled by server.

Syntax

```
nCount = objServer.PendingMessagesCount
```

Return values

Integer

number that indicates the current depth of the incoming messages queue.

Remarks

Server object creates the thread for handling network messages asynchronously. While Director's main thread is running your Lingo code, server's thread may receive incoming messages and send outgoing messages. Incoming messages are placed into the queue before they will be handled by Lingo in the main Director thread. The current depth of this queue is returned by this property.

Port

Returns the port number, on which the server is running.

Syntax

```
nPort = objServer.Port
```

Return values

Integer

Port number used to start server.

Connections

Returns the list of addresses of the currently connected clients.

Syntax

```
lstAddresses = objServer.Connections
```

```
nCount = objServer.Connections.Count
```

Return values

List

Linear list of strings with client addresses currently connected to the server.

Remarks

Client's address is returned as a string in a form: "IP:Port".

Data

Returns the contents of the currently processing message.

Syntax

```
valData = objServer.Data
```

Return values

```
valData
```

Lingo value passed to the server as a message contents. See [supported data types](#) for details of what type can be this Lingo value.

RemoteTierAddress

Returns the address of the client whose message is being processed.

Syntax

```
strAddress = objServer.RemoteTierAddress
```

Return values

```
String
```

Client's address whose message is being handled in a form: "IP:Port".

Subject

Returns the subject of the currently processing message.

Syntax

```
strSubject = objServer.Subject
```

Return values

```
String
```

Subject of the message being handled in a form: "IP:Port".

Remarks

Subject is used by MUS protocol. It is specified by client [sending](#) message. Replying to the message keeps the message subject unchanged.

SenderId

Returns the SenderId of the currently processing message.

Syntax

```
strSenderId = objServer.SenderId
```

Return values

```
String
```

SenderId of the message being handled in a form: "IP:Port".

Remarks

SenderId is used by MUS protocol. It is specified by client when it [connects](#) to the server.

Synchronized Client Object

Synchronized client object is the simple client side for the communication with the Server object over Internet. Similar to multiuser xtra, it can send messages to the server and receive what it replies. Since it uses MUS protocol, it is possible to [use multiuser xtra](#) to connect to the server part of SocketServerXtra.

To create synchronized client object use xtra-level [New](#) method:

```
objClient = xtra("SocketServerXtra").New( #SyncClient )
```

Use [Interface\(\)](#) method to get the short description of methods and properties provided by the object.

Use [Connect\(strServer, nPortNumber, strSenderId \)](#) method to establish connection to the specified server.

Once connection is established you may communicate with the server via [Send\(valData \)](#) method. Synchronized client is intended for use in remote method call scenario, thus `Send(valData)` returns only when the client has received the server's response. This scenario goes well for local area network (LAN) applications, although it may be used in wide area network (WAN) applications too in some cases.

Using native multiuser xtra allows connect to SockerServerXtra's server in message oriented manner (asynchronous client).

Methods

Interface()

Returns the short description of methods and properties provided by the object.

Syntax

```
strDescription = objServer.Interface()
```

Return values

String

Returns a string with short listing of methods and properties exposed by this object.

Remarks

This method does not clear the last error flag. It means this method does not affect the last error information for the particular object.

Sample

The sample shows how to get create a new object instance and view its interface description.

```
net = xtra( "SocketServerXtra" )
objConnection = net.New( #SyncClient )
put objConnection.Interface()
```

Connect(strServer, nPortNumber, strSenderId)

Establishes the connection to the specified server passing optional `SenderId` string.

Syntax

```
objConnection.Connect( strServer, nPortNumber, strSenderId )
```

Parameters

strServer

String with the URL or IP or network name of the server.

nPortNumber

Integer port number matching the port used to start server.

strSenderId

Optional string with the user name. It is used by MUS protocol, but it is optional to SocketServerXtra. Client object keeps the `SenderId` and sends it with every message to server. Server side movie can access this string via [SenderId](#) property.

Sample

The code below shows how to ask server for something:

```
net = xtra("SocketServerXtra")
objConnection = net.new( #SyncClient )
-- Connect to the server
objConnection.Connect( "server_name_or_ip", 4545, "UserName" )
-- Prepare data to be sent
valSomeLingoValue = [ 1, 2, "3", [1, 2], [#propName:propValue], #etc ]
-- Sending data and waiting while server replies
valServerReply = objConnection.SomeSubject( valSomeLingoValue )
-- Seeing what server has replied
put valServerReply
```

Send(strSubject, valData)

Sends the specified data to the server and returns what server replies

Syntax

```
valResult = objConnection.Send( strSubject, valData )
```

```
valResult = objConnection.Subject( valData )
```

Parameters

strSubject

String that identifies the subject of the message to be send to the server. Subject is used by MUS protocol. Server side movie can access this string via [Subject](#) property.

valData

Lingo value to be passed on server. Server side movie can access this string via [Data](#) property. See [supported data types](#) for details of what type can be this Lingo value.

Returns

valResult

Lingo value passed back as a server reply. Server side movie sets this data via [Reply\(valData \)](#) method. See [supported data types](#) for details of what type can be this Lingo value.

Remarks

Synchronized client object blocks the Director main thread until server replies or connection is closed. Programming in the synchronized mode is much easier than supporting asynchronous mode. For some network applications synchronized mode may fit well, others may require asynchronous mode, when you send message and your application continues running while reply is coming back to you. Use native multiuser xtra as [asynchronous client](#) to SocketServerXtra server's object.

Calling other methods

Other methods are automatically mapped to [Send](#) method with appropriate subject. So calling: `cnn.SomeMethod(valData)` is the same as:

```
cnn.Send( "SomeMethod", valData )
```

This mapping may simplify programming network applications by simulating remote procedure calling. Client side calls normal Lingo method and receives normal Lingo return value, while internally network message is generated and passed to the server and back.

Properties

Server

Returns the address of the server used to open connection.

Syntax

```
strServer = objConnection.Server
```

Return values

String

Address of the server used to open connection.

Port

Returns the port number used to open connection.

Syntax

```
nPort = objConnection.Port
```

Return values

Integer

Port number used to open connection.

Xtra-level methods

Init(nDebug)

Performs generic xtra initialization. Allows setting debugging mode as a default for newly created objects.

Syntax

```
bSuccess = xtra("SocketServerXtra").Init( Integer nDebug )
```

Parameters

nDebug

Optional. Debugging mode for newly created objects. This parameter can be one of the following values.

Value	Meaning
0	No debugging support. Release behavior.
1	Simple debugging. Any error is automatically printed in Messages window.
2	Advanced debugging. When any error is occurred, the xtra calls movie level handler <code>SocketServerXtra_DebugEvent(strMes, nCode)</code> .

Return values

If the method succeeds, it returns true. Otherwise it returns false.

Remarks

Every object created by SocketServerXtra can detect errors returned by wrapped objects. Internal SocketServerXtra errors (type casting problems etc) could happen too. Normally these errors could be trapped programmatically by checking object last error status after any meaningful call to the object. See [error handling](#) and [debugging](#) support for more details.

Note: This method does not affect objects that already exist at the time of calling this method. You may use [DebugMode](#) property to change the debugging mode of the particular object directly.

New(symObjectType)

Creates a new instance of the object of the specified type.

Syntax

```
objSocket = xtra("SocketServerXtra").New( Symbol symObjectType )
```

Parameters

symObjectType

Symbol value specifying which object to create. This parameter can be one of the following values.

Value	Meaning
-------	---------

#Server	Server object
#SyncClient	Synchronized client object

Return values

Object

If object is created successfully the method returns the new instance of SocketServerXtra object of the specified type.

String

If the xtra failed to create requested object the method returns a string with error description.

Sample

This sample shows basic server functionality:

```
property m_objServer
on beginSprite me
    net = xtra("SocketServerXtra")
    m_objServer = net.new( #Server )
    -- Start server bound to the port 4545
    m_objServer.Start( 4545 )
end
on exitFrame me
    -- Read available incoming messages one by one
    repeat while objServer.ReadNext()
        -- Read what is passed
        valIncomingData = oServer.Data
        strSubject = oServer.Subject
        -- Prepare reply based on incoming data
        valOutcomingData = valIncomingData
        -- Reply to the message
        oServer.Reply( valOutcomingData )
    end repeat
end
```

The code below shows how to ask server for something:

```
net = xtra("SocketServerXtra")
objConnection = net.new( #SyncClient )
-- Connect to the server
objConnection.Connect( "server_name_or_ip", 4545, "UserName" )
-- Prepare data to be sent
valSomeLingoValue = [ 1, 2, "3", [1, 2], [#propName:propValue], #etc ]
-- Sending data and waiting while server replies
valServerReply = objConnection.SomeSubject( valSomeLingoValue )
```

```
-- Seeing what server has replied  
put valServerReply
```

Version()

Syntax

```
strVersion = xtra("SocketServerXtra").Version()
```

Return values

String

Version string in a form of 5 point delimited items: "SocketServerXtra.1.0.0.4".

The first item is the xtra's name "SocketServerXtra".

The second item is the major xtra's version.

The third item is the subversion number. It indicates noticeable changes.

The fourth item is the minor version number. It indicates minor changes.

The last item is the absolute build number. It is auto incremented with every release build of the xtra.

Data types supported

The current version supports following Lingo data types:

- Integer or Float number
- String (including strings with zero bytes inside)
- Symbols
- Linear list of supported Lingo values
- Property list of supported Lingo values

Here is the example of supported values:

```
-- Data to be sent  
valSomeLingoValue = [ 1, 2.0, "3", [1, 2], [#propName:Value], #etc ]  
valSomeLingoValue = "Just a string"
```