# Conga 3.0 Upgrade Guide

Version dated May 12th, 2017.

## Introduction

At the time of releasing Dyalog APL version 16.0, these Release Notes are the only complete set of documentation available for the new features of Conga version 3.0. The content of these Release Notes are being integrated with the existing Conga documentation, and will be made available shortly after the release of Dyalog v16.0.

The revised Conga manual will appear on http://www.dyalog.com/documentation\_160.htm when it is complete. Contact [support@dyalog.com](mailto:support@dyalog.com) for more information if necessary.

## New Features

Conga version 3.0 provides a large number of new features designed to make it easier to write network-based applications:

1. **Multiple Isolated “Roots”:** Make it possible for separate components running in the same Dyalog process to use Conga without interfering with each other in any way.
2. **New Server modes:**
   1. *FIFOMode*: Processed messages in strict chronological order (does not allow filtered queries on individual connections)
   2. *ConnectionOnly*: The server object will not receive data – the application is expected to spawn individual APL threads to handle each connection object separately.
3. ***Timeout* and *Disconnect* as events:** Instead of error codes 100 and 1119, timeouts and disconnections can optionally be signalled as events.
4. **Temporarily prevent new connections** by setting the *Pause* property.
5. ***Sent* event:** Avoid flooding buffers with un-transmitted data by requesting a *Sent* event when the last byte has been transmitted.
6. **Transmit files** without reading them into APL first.
7. **HTTP protocol support:** Rather than receiving text blocks and deciphering them, you can elect to receive *HTTPHeader*, *HTTPBody*, *HTTPChunk*, *HTTPTrailer* events after Conga has parsed the incoming messages.
8. **WebSocket support**: HTTP connections can be upgraded to bi-directional web sockets, allowing asynchronous bidirectional data transmission.
9. **Positive and Negative lists of Peer address ranges:** Support for applications which only wish to service connections made from certain locations.
10. **A numeric *Version* property:** Makes it easier to write applications that need to know which version of Conga they have available.
11. **Support for GnuTLS 3.4.16.**
12. **Dynamic loading of secure socket support:** If you do not use secure sockets, you do not need to install or ship the secure part of Conga with your application.
13. **Simpler configuration:** By default, Conga libraries are always loaded from the folder where the interpreter executable is located, bypassing the need for LIBPATH or similar environment variables.
14. **Shared Unicode/Classic library:** The same .so or .dll file is used by Classic and Unicode versions of Conga (32 and 64 bit are still separate).
15. **Experimental UDP support:** Undocumented; contact [support@dyalog.com](mailto:support@dyalog.com) for more information.
16. **Numerous new samples and tools.**

## 0. Compatibility and Code Distribution

Conga 3.0 is designed to be compatible with Conga 2.x, but the APL code that wraps the Conga libraries has been substantially rewritten in order to provide support for multiple isolated roots, and to provide simpler ways to create client and server applications. To take advantage of some of the new features, you will need to make changes to the way that you initialise Conga-based applications.

In order to provide a smooth migration path from earlier versions, and allow you to take new features on board when you are ready, a number of workspaces are shipped with Dyalog 16.0:

|  |  |
| --- | --- |
| **Workspace** | **Description and Comments** |
| conga.dws | Contains a new (but upwards compatible) version of the old DRC namespace, and the new Conga namespace. It is provided in order to allow existing code which uses ('DRC' ⎕CY 'conga') to continue to work. Also contains a reduced set of the most common examples in the Samples namespace. |
| conga3.dws | Contains only the new Conga namespace; this is the recommended workspace to use for new applications. |
| conga2.dws | This workspace is provided as a fall-back in case of problems with the new APL code – may be withdrawn in Beta testing does not reveal problems. Contains old code which simply loads the Conga v3.0 DLLs (this is the workspace that was provided under the name conga.dws in pre-Beta versions of Dyalog version 16.0). |

Existing Conga 2.x-based applications typically initialise Conga by calling (DRC.Init '') . Unless you wish to make use of the multiple-root feature described in the following, we recommend creating a default instance of Conga 3 as follows:  
  
 iConga←Conga.Init ''

The above expression connects to the Conga root object named DEFAULT, which is shared by all applications which use an empty right argument to Init. If you name the instance variable DRC, existing application code which refers to DRC (except for the call to DRC.Init) should work unchanged. In other words, if you replace the expression (DRC.Init '') with (DRC←Conga.Init ''), and ensure that you have the Conga namespace rather than DRC loaded into your workspace, the rest of your Conga 2.x application should run unchanged.

The code samples that used to reside in the Samples namespace in earlier Conga workspaces have been withdrawn, because better alternatives now exist. A cut-down set of the most widely used samples can be found in the Samples namespace in conga.dws. A new set of samples can be found in the folder Samples/Conga, and tools such as the new HttpCommand which is intended to replace the old Samples.HttpGet, can be found in Library/Conga.

## 1. Multiple Isolated Roots

As the use of Conga has grown, it has become common for more than one component in an application to use Conga, leading to potential name conflicts between client and server objects created, and clashes between different state settings. In particular, when an application and the tools used to maintain it both use TCP communications, they need to be independent of each other. Each may need to be restarted or reset without interfering with the other. Version 3.0 allows each component to have its own “Root” under which Conga objects are created, and perform operations like deleting all existing Conga connections in order to restart, without fear of interference with other components.

In versions before 3.0, all components in the same process used the same DRC namespace: The first user would call DRC.Init and received a clean return code; subsequent calls to Init warned that Conga was already initialised. If a component decided to reset or re-initialise Conga, all other components would be affected by this.

In version 3.0, the DRC namespace is still provided for backwards compatibility, but – as mentioned in the previous section - it is recommended that you use the Conga.Init function to create or select a specific named root for your application or component, rather than calling DRC.Init. For ad hoc use, you can use an empty right argument to connect to the DEFAULT root, but for an application which may need to manage the state of Conga, we recommend using a right argument to identify your application.

iConga←Conga.Init 'MyApp'

The above statement can safely be called anywhere in your application; if a root with that name already exists, a reference will be returned to the existing root instance. If you want to be sure that a new instance is created, you can use Conga.New. In this case, an empty argument will generate a new unused root name, and a non-empty argument will signal an error if the root name is already in use. The function RootNames can be used to get a complete list of existing roots:

iC1←Conga.Init '' ⍝ Use the default root

iC2←Conga.New '' ⍝ Create a new one with a generated name

Conga.RootNames

DEFAULT IC1

To re-initialise your root, simply erase the reference (all references) to the instance; it will be cleaned up, and you can create a new one.

**Warning:** The intention is NOT that you create a large number of roots. The process of creating and tearing down roots is expensive and complex. Components may need a separate root, but you should not create new roots in order to make ad hoc queries on the internet. In this case, use the DEFAULT root that you can get a reference to by passing an empty argument to Conga.Init.

## 2. New Server Modes

The default mode for a server allows an application to selectively wait on the entire Server, receiving both connection-related events and data transmissions to all or part of the tree of objects that make up the server. As usage patterns evolve, Conga is also evolving to provide modes which are better suited to, or tuned for these patterns.

### 2.a FIFOMode

For high volume services with hundreds or thousands of connections, the cost of providing filtering functionality becomes unacceptable. In addition, the filtering mechanism could lead to certain connections receiving better service than others. The FIFOMode switch turns off the ability to call the Wait function on a subset of a Server object hierarchy. In return, you get significantly less CPU consumption, and are guaranteed that messages come off the queue in strict chronological order of arrival.

Enable *FIFOMode* for a server using SetProp:  
  
 iConga.SetProp 'S1' 'FIFOMode' 1

Once *FIFOMode* is enabled, attempts to Wait on a connection object which is a child of the server will fail with error 1142 ERR\_FIFOMODE.

### 2.b ConnectionOnly

Some server applications have a structure which makes it convenient to launch an APL thread per client connection, and leave that thread running for the duration of that client session. The *ConnectionOnly* switch enables a mode where Wait on the server object will only ever report Connect events: Individual application threads are expected to call Wait on the connection that they are managing.

Samples/RPCServices/ThreadedRPC contains an example of a server which uses ConnectionOnly and runs a thread per connection.

## 3. *Timeout* and *Close* as Events

Many application developers have found it inconvenient that “normal” events such as a timeout due to inactivity, or the closing of a connection, are reported as if they were errors with non-zero return codes fro Wait, rather than being classified as events.

In version 3.0, it is possible to receive events but, as this is a breaking change, it is not enabled by default: you need to set the *EventMode* property to enable it. **Dyalog strongly recommends that you enable *EventMode,* and is likely to make it the default in a future version of Conga.**

Following:

iConga.SetProp 'S1' 'EventMode' 1

Return codes 100 and 1119 will no longer issued, instead Wait will return one of:  
  
0 S1 Timeout  
0 S1 Close

## 4. Temporarily Prevent New Connections to a Server

If a server needs a break from incoming connections, either because it is preparing to shut down for maintenance, is overloaded, or has some other reason to need a break, it is now possible to achieve this using the *Pause* property, which has three possible settings:

|  |  |
| --- | --- |
| Setting | Effect |
| 1 | Keeps the listening socket open but does not accept new incoming connections. Connection attempts which have not timed out on the client side will be accepted when Pause is set to 0. |
| 2 | Closes the socket but keeps the server object alive, when Pause is set to 0 the socket will be re-created. |
| 0 | Resume normal operations. |

For example:

iConga.SetProp 'S1' 'Pause' 1 ⍝ Do not accept connections

## 5. *Sent* event

If you are transmitting a large amount of data in chunks, Conga allows you to make repeated calls to the Send function without waiting for the previous send to complete. This can cause large amounts of data to accumulate in buffers either in Conga or the network layer, which may be undesirable.

In Conga 3.0, you can request a receipt upon the completion of the actual transmission, by appending a 3 following the data passed to the Send function:

iConga.Send 'C1' data 3  
0  
 iConga.Wait 'C1'  
0 C1 Sent 0

When using command mode, the sent event will be overridden if by the answer on a command, if the response to a commend arrives before you enter the next Wait, you will simply get the response and the Sent event will be suppressed.

## 6. File Transmission

When an APL-based server needs to transmit the entire contents of a file, earlier versions of Conga have required that you first read the contents of the file into the APL workspace and then pass it to the Send function - a process which is obviously inefficient.

For a connection which is not in *Command* mode, Send now accepts a two-element nested vector: The first element is data to be transmitted first, and the second element contains a file name, the contents of which will be transmitted after the initial data. The first element allows you to prepend header information to the transmission, where necessary. For example:

iConga.Send 'C1' ('' 'c:\mywebsite\index.html')  
0

## 7. HTTP Protocol Support

A common use of Conga is to act as an HTTP client, retrieving data from web sites or making web service requests – or as an HTTP server, serving up data managed by an APL application. In earlier versions, this required buffering data and parsing the HTTP protocol in APL.

In version 3.0, you can set the Mode of any Client or Server to be HTTP. If you do this, the normal Receive and Block events are replaced with events that signal the arrival of a complete piece of HTTP protocol: HTTPHeader, HTTPBody, HTTPChunk and HTTPTrailer. This not only simplifies the task of receiving HTTP data, it also significantly improves performance my moving the parsing into multi-threaded, asynchronous C code dedicated to this task.

At present, Conga does not provide any further processing, such as taking headers apart or decoding base-64 encoded data; this may follow in future releases.

When transmitting data, you are required to generate valid HTTP messages. The only support that Conga 3.0 provides is to add a valid Content-Length header when transmitting a file (see File Tranmission).

## 8. Web Sockets

An established HTTP connection can be upgraded to bi-directional websocket connection, which allows both client and server to transmit data at any time, rather than sticking to the normal cycle of the client making a request, followed by a server response.

### Web Socket Upgrade - Client Side

It is the client that requests the upgrade. To upgrade a Conga-based client, set the *WSFeatures* property to whether you wish to automatically accept a positive response from the server (1), or you need to validate the response and confirm it (0). Unless you know something about websocket internals, auto-upgrade is highly recommended -   
  
 iConga.SetProp clt 'WSFeatures' 0 ⍝ Do not automatically accept  
0

Next, set the *WSUpgrade* property to a three-element vector containing a URL, hostname (normally the same one you already connected to) and any necessary header information that the particular server you are connecting to may be looking for in order to decide how to handle the connection. For example:

iConga.SetProp 'C1' 'WSUpgrade' ('/' 'localhost' 'some-setting: value')  
0

Now call Wait, and if the server accepts your request the response will be a *WSResponse* event, with data containing header information that the server has decided to send:

res

0 C1 WSResponse HTTP/1.1 101 Switching Protocols  
 Upgrade: websocket

Connection: Upgrade  
 Sec-Websocket-Accept: G/cEt4HtsYEnP0MnSVkKRk459gM=

If *WSFeatures* had been set to 1, the response would simply have been a WSUpgrade event:  
  
0 C1 WSUpgrade 0   
  
If auto-accept is not enabled, we now need to examine the headers, decide whether they are OK, and finally set the *WSAccept* property:

iConga.SetProp 'C1' 'WSAccept' ((4⊃res)'')  
0

You are required to confirm the headers that you wish to accept as the first element, the second element is not used but required for symmetry with the server call (see the next section). The next call to Wait should return a WSResponse event, after which the socket can be used as a websocket.

0 C1 WSResponse HTTP/1.1 101 Switching Protocols   
 Upgrade: websocket   
 Connection: Upgrade

Sec-Websocket-Accept: G/cEt4HtsYEnP0MnSVkKRk459gM=

### Web Socket Upgrade - Server Side

When a client requests a web socket upgrade, the server will either receive a *WSUpgrade* (if it has WSFeatures set to 1) or a *WSUpgradeReq* event from a call to Wait:

0 S1.CON00000000 WSUpgradeReq GET / HTTP/1.1  
 Host: localhost   
 Upgrade: websocket  
 Connection: Upgrade  
 some-setting: value  
 Sec-WebSocket-Version: 13  
 Sec-WebSocket-Key: KSO+hOFs1q5SkEnx8bvp6w==

The format of the messages is the same; in the event of a *WSUpgrade* the 4th element is for your information only and the upgrade has been done, otherwise you need to follow a similar pattern as the client, either closing the connection if the request is denied, or responding with a confirmation of the received headers plus any information that need to be sent to the client:  
  
 iConga.SetProp 'S1.CON00000000' 'WSAccept' ((4⊃res)'server-says: hello')  
0

At this point, even before the client has received a *WSResponse* (if it is a Conga client, or the equivalent in JavaScript or some other programming language), the socket is considered open and the server should be able to transmit data. However, it may be prudent to wait for the client to initiate communications, as final confirmation that the websocket is operational.

Once an upgrade to websocket has happened, incoming data on the socket will be delivered in the form of a WSReceive event, which is accompanied by a 3-element vector containing *data* in the 1st element, followed by a Boolean “*final*” flag which indicates whether this is the last transmission in a series, and an *opcode* indicating the data type. The opcode can be 1 for UTF-8 Text or 2 for Binary data (represented as an integer vector with values between ¯128 and +127). The opcode can also be 0 to indicate a “continuation”, in which case the data type is expected to be the same as the previous message.

iConga.Wait 'C1'

┌─┬──┬─────────┬─────────────────┐

│0│C1│WSReceive│┌───────────┬─┬─┐│

│ │ │ ││Hello World│1│1││

│ │ │ │└───────────┴─┴─┘│

└─┴──┴─────────┴─────────────────┘

To transmit data on a websocket, use the send function and pass a 2-element vector containing the *data* and *final* flag, and an optional opcode. If you do not provide opcode it will be set according to the data type:

iConga.Send 'C1' ('How do you do?' 1)  
0

The use of the Send function as described above should cover all common uses of websockets. If you have special requirements, you can also use an argument which is an integer vector of values between ¯128 and +127, and send a pre-formatted websocket message containing all data and control fields and blocks. This document makes no attempt to describe the low-level websocket protocol, if you don’t already know what to do we recommend you steer clear of this option!

## 9. Positive and Negative Lists of Peer Address Ranges

Conga 2.7 allows a server to know the “Peer Address”, which identifies the other end of a connection. If you need to restrict access to a server to certain ranges of addresses, it was possible to check the address and close the connection.

Version 3.0 simplifies this process by allowing a server to declare a set of endpoint ranges that should either be allowed or denied. For example:

ipv4←⊂'IPv4' '192.168.202.1/24'

ipv6←⊂'IPv6' 'fe80::d189:fd4:7003:a0a3/120,fe80::9df3:f956:84f5:12ab/120'  
 iConga.Srv 'S1' '' 5123 'Text' ('AllowEndpoints'(ipv4 ipv6))

0

Each list of endpoints is a 2-element vector, the first element must be IPv4 or IPv6, and the second element a list of addresses and range sizes, separated by commas. The *DenyEndpoints* property accepts data in the same format, and allows you to set up address ranges from which connections will be rejected.

If both *AllowEndpoints* and *DenyEndpoints* are specified at the same time and a connection is made from an address which is in the intersection of the ranges, the connection will be rejected.

## 10 A numeric Version Property

To make it easier to write code which adopts different strategies depending on the version of Conga which is available, Version is reported as a three element vector containing the major and minor version numbers, followed by the SVN revision number that the current version of Conga was built from:

iConga.Version  
3 0 1215

## 11. Support for GnuTLS 3.4.16

Conga 3.0 integrates GnuTLS 3.4.16 for secure communication. Most of the improvements are invisible to the end user, but provide more secure and more efficient secure connections. The new version of GnuTLS can use certificates in the Windows certificate store directly, and contains many improvements, making Conga more able to connect to servers which new security features.

## 12. Dynamic loading of secure socket support

Earlier versions of Conga loaded the TLS library on startup, and failed if it was not possible to locate the library. Conga 3.0 only loads the secure support library when the first secure feature is used. This means that you do not need to ship the secure library (named conga30sslbb) if your application does not make use of secure sockets. If the library is not present and secure features are used, an error will be reported.

## 13. Simpler Configuration

Conga is no longer dependent on LIBPATH or other environment variables which are used to specify the location of dynamically loadable libraries. The APL interpreter will always look for the Conga library in the folder that the APL interpreter was loaded from, and Conga loads the secure socket library from the same location that it was loaded from.

It is still possible to specify the exact location of the Conga library when initialising Conga within the APL application. This requires the secure socket library to be in the same location, if secure features are used.

## 14. Shared Unicode/Classic library

The same .so or .dll file is used by the Classic and Unicode versions of Conga. (32 and 64 bit are still separate). Not sure this is worth mentioning in the user documentation.

## 15. Experimental UDP Support

Experimental UDP features have been prototyped, but the features have not reached a level of maturity that warranted documenting in the 3.0 Upgrade Guide. If you have a need for UDP functionality, and would like to participate in ongoing design and testing of these features, please contact [support@dyalog.com](mailto:support@dyalog.com).

## 16. Numerous New Samples and Tools

The folders library/conga and samples/conga, below the main Dyalog folder, contain a number of new tools and samples. The code is also available on line in repositories named library-conga and samples-conga under <https://github.com/Dyalog>.

[would be nice with some lists and descriptions here]