

# Citect S-Business

Server Technology and Redundancy in Citect

**Technical Paper**

**December 2002**



## **Abstract**

This document is an attempt to educate the reader on some of the more obscure operational facts of a networked Citect system. It deals with server redundancy from the IO point of view, as well as touching on the performance of alarm, trend and report servers. There is also a brief discussion of Citect's internet capability.

## **About Citect**

Citect is a leading, global provider of industrial automation, real-time intelligence, building automation, next generation manufacturing execution systems (MES) and on-demand benchmarking applications. Leveraging open technologies, CitectHMI/SCADA, Ampla, Nexa and Meta connect to multiple plant and business systems. Its products are complemented by Professional Services, Global Customer Support and Educational Services. Citect solutions are installed in over 80 countries and implemented in numerous industries: mining, metals, food and beverage, manufacturing, facilities, water, gas pipelines, power distribution and pharmaceuticals. Headquartered in Sydney Australia, Citect has representation in Oceania, Southeast Asia, China and Japan, North and South America, Europe, Africa and the Middle East.

[www.citect.com/support](http://www.citect.com/support)

## **Contacts**

customerservice@citect.com

## Contents

### The Citect IO Server

#### Regular IO

Any and every Citect machine is a display (or data) client. Whether or not they are designated as servers, they are still clients. Put another way, all Citect machines are at least users of system data. This point should be considered closely when analysing the workings of a Citect system. All requests for data originate from the client, and of course the client is the ultimate destination for system data. As far as networking goes, Citect clients don't talk directly to any IO device, they all talk to servers. A client (an IO client especially) does not necessarily talk to the local server – if the client is after data primarily sourced from a separate server it will go to that server, regardless of whether the local server can access the device in question. Conceptually therefore, it is more accurate to consider each Citect node as two distinct modules, a server and a client that don't necessarily have much to do with one another.

At this point it seems appropriate that some comment be made on terminology. In many systems there are two IO servers, the users of which call them primary and standby. However, this naming convention is not really appropriate in many cases, since quite often one server is primarily responsible for some IO devices, while the other is primarily responsible for the others. Which is primary? Well, neither. Neither is there a specific standby server for that matter, since each can cover for the other if either goes down. It seems more accurate therefore to regard IO redundancy in terms of channels, where a channel is defined as a path from a data source (IO device) to a data sink (client). When consideration is given to how IO devices are defined in a Citect project, it makes much more sense to think of things this way. After all, it is possible to use a single server and two connections to the same device to provide some redundancy. The same server would therefore be both the primary and the standby! Also, in the case of IO device redundancy (having two PLCs in case one fails) it is possible to have one server connected to both, but only one of these devices can be defined as primary. As a result, the term "channel" will be used (in this document) and encouraged, where a channel is a device definition in a Citect project. "IO device" definitions in Citect don't merely define IO device characteristics, they also define how a client can get data from the device – a channel to the IO. The server responsible for the primary channel to an IO device will be known as the primary channel server. Other channels to a particular device (or data image) will be referred to as standby channels. The server(s) responsible for the standby channel(s) will correspondingly be called the standby channel server(s). A single Citect IO server can be both a primary channel server and a standby channel server at the same time. The "primary" and "standby" designation is with reference to the IO device in question and is not an absolute machine label.

## Simple Redundancy Schemes

The easiest way to understand redundancy in Citect is from the client's point of view. Essentially, the client gets to choose which channel to use to the device in question, understanding first that it always prefers the primary channel. If the client is not getting satisfaction from the primary channel then it may at that point decide to change to the standby (or a standby, if there are more than one). The other thing to be aware of is that Citect tracks data sources using device numbers. Citect groups together channels that have the same device number, and regards them as a single data image. The compiler prepares requests for tags (to be used on pages, alarms, trends etc) and maps them to a device number. This device number is considered at runtime and then a request is made to one of a set of channels which (potentially) service that number. So while the people configuring Citect tags, servers and devices might think in terms of tagnames and PLC names, Citect runtime uses addresses and device numbers. Quite often primary and standby channels will be given the same name (aka IO Device name) because those configuring the project are thinking in terms of ultimate destinations (the box where the request ends up). As a result when tags are being configured, the IO Device drop down list offers a choice which contains numerous instances of the desired IO Device. Which one should be picked? Many users would pick the top one, reasoning that it is probably the primary device and that is the one they would prefer. Well it doesn't matter which one is picked, since they both have the same number and that is what counts to the compiler.

## Server Redundancy

The most basic case of IO redundancy is that where two IO servers are simultaneously connected to a single physical PLC. One IO channel (IO server – PLC combination) is defined to be primary, and the other is standby. All clients running this project will therefore prefer to acquire data from the primary channel, and will otherwise defer to the standby channel. Even the client sharing a machine with the standby channel server will follow this pattern of operation.

There are a number of situations that this kind of redundancy deals with. They are when:

- 1 The primary channel server is shut down. As a part of the shutdown procedure the server notifies all IO clients that it is shutting down. The clients are then free to gracefully close their sessions to the primary channel server and commence transactions on a standby channel. This notification type method provides a bumpless changeover from one server to the next.
- 2 The primary channel server fails and the client therefore loses communications with it. Requests to the primary channel server will consequently timeout eventually and the client will raise an alarm indicating the server cannot be contacted. The client will then direct requests to the standby channel server and continue operation. When the

primary channel server comes back online it will notify clients of its' availability and normal operation will recommence ie. clients will start making requests of the primary channel server once again.

3 The network connecting the client to the primary channel server has failed. From a client point of view this situation is identical to case (2) and the clients treat it in the same way. Requests are directed to the standby channel server rather than the primary. However, in the previous case the server notified clients of its' availability (using a network broadcast) when it restarted. This time though the server has not failed and is not therefore inclined to advertise itself as a result. To handle this situation, when comms are lost between client and server the clients periodically call the primary channel server in an attempt to reconnect.

4 The comms between the primary channel server and the IO device (as defined by the primary channel) have failed. At this point the device is declared offline and the clients are warned as such. Clients then switch to a standby channel. The primary channel server meanwhile continuously tries to re-establish comms with the offline device. When the device comes back online the server notifies the clients that the primary channel is once more available and the clients begin using it again.

### **Data Path Redundancy**

Data path redundancy is essentially the provision of multiple connections between the PLC and the IO server. For example, a PLC might be connected to the IO server via ethernet, and have a backup serial link for use in emergencies when the ethernet is unavailable. From Citect's point of view, there are two data channels defined, one primary (ethernet) and the other standby (serial). These are both defined to use the same server, naturally. This trick of using substantially different communications methods for the two paths is only possible where the Citect drivers allow it. The ethernet driver in this case may have to be detuned (to the serial driver level) to allow the compiler to prepare for redundant comms. Alternatively, if the same communication method (and thus driver) is used on both channels this issue does not come into consideration.

### **IO Device Redundancy**

IO Device Redundancy is implemented by duplicating the device itself, be it PLC or some other kind of box. Essentially this looks to Citect the same as the previous example, that of Data Path Redundancy. It is configured the same way, using two channels, one primary and the other standby. The trick with IO Device redundancy is in the management of the device changeover, which is not normally within the scope of Citect. A simple case may be a master and a slave PLC, where the master continually updates the slave's memory (using perhaps a separate comms bus) to keep it up to date. However, the reader should be advised that it is possible that Citect may end up writing control values to the wrong PLC in a redundant pair. This may happen if the comms cable the primary channel depends upon fails, yet the master PLC remains operational. As far as Citect is concerned, the master PLC is offline, so it switches to the standby channel, which corresponds to the slave PLC. At this point however the PLC is not in a position to act on the information, since as a slave it's memory is essentially not it's own. It may be necessary to implement some clever data monitoring arrangement in Citect;

complete discussion of the perils of successfully implementing such a system is not really within the scope of this document and will not be discussed further here.

### **Advanced Redundancy Schemes**

Advanced or complex redundancy schemes comprise some elaborate combination of the simple schemes detailed above. In most cases the net result from a configuration point of view is simply extra channel definitions, giving clients more choices for the delivery of data in a timely fashion. For instance, combining IO server redundancy with IO device redundancy doubles the number of channels defined and can get quite mind boggling if pursued to subsequent levels. In the same way, ensuring IO Server redundancy by using multiple standby channel servers (rather than just one) has the same effect, adding extra channels for clients to choose from. Although Citect will work at these extremes, the reader should be aware that multiple standby channels are not the norm and some unpredictable behaviour may be encountered. It's not that the redundant functionality won't work, however there is a chance that clients may choose a channel other than that preferred by the designer. In general, channels should be defined in the order in which they are preferred. Position in the IODevices definition table should be followed. Extra code (using IODeviceControl) may be required to manually disable standby channels and therefore ensure correct operation.

### **Disk and Memory IO**

Disk IO is different from external IO in that the data image is usually stored on the hard drives of the servers, rather than being separate. To ensure the system can protect Disk IO then it must replicate the Disk IO data from server to server. Citect does this by replicating all writes to the Disk device from the primary channel server to the standby, apart from the clients. For example, when a client performs a write on a disk device, it will naturally direct the operation to the primary channel. The primary channel server will process this request and at the same time issue a write operation to the standby channel so that the standby image is kept up to date. Then when clients call on the standby channel server for data from the disk device for whatever reason it will be current. This mode of operation is called StandbyWrite. Another consideration is that the system be able to "catch up" on any data changes which might have occurred while a server was down. To accomplish this, the disk data files associated with Disk IO devices are cross-copied (new copied over old using file stamp checking) at server startup to ensure files are up to date initially. This is the reason for the dual address format in a disk device definition. From a client point of view there is no difference between Disk IO and external IO; they are both treated the same way and data is available from either everywhere on the network.

Memory IO is machine specific. When it comes to memory IO, each Citect node is like it's own IO server and IO device rolled into one. Memory IO cannot be shared between machines like Disk IO. Memory IO channels are not pertinent to IO servers – regardless of the server definition in a Memory IO channel, every

Citect machine running that project gets its own copy of the Memory IO device.

### Internet and Proxy

The Internet Server in Citect contains functionality to enable the quick and painless setup of a Citect networked system over the internet or intranet. It is essentially an FTP server added to Citect's normal feature set. In addition to this, there are two notable new technologies. One is the IO Proxy server, which allows internet clients access to the entire range of IO servers via a single point of contact. The proxy server can also act as a request concentrator, so extra clients will not add so much extra traffic to the Citect network. The other technology now available is TCP sockets, which Citect IDCs use to talk across the internet. In fact, all Citect nodes can now use sockets instead of NetBIOS based networking.

Redundancy on an Internet-based Citect system is compromised to some extent. It is not possible to have a hot standby arrangement for Internet servers, although clients could choose to connect via a different Citect Internet server once the initial choice has failed. This is currently done manually however. Similarly, there is no fail-over possibility for proxy servers. Also, IDCs cannot connect to more than one report/alarm/trend server without connecting into the Citect network proper ie. there is no proxy available for report/alarm/trend data.

### The Citect Data Server

Citect data is that which Citect produces, rather than extracts from PLCs and so on. Alarms, Trends and Reports are examples of Citect data. A traditional networking arrangement will see these data servers in pairs, and a group of pairs is referred to as a cluster. A cluster of data servers may be arranged on a large group of machines, or it may be on a simple pair. Clusters are as much about supporting each other as they are about providing redundancy to clients. For example, a pair of alarm servers not only allow redundancy from the clients' point of view of the system but also back each other up to ensure smooth operation.

At this point, a bit of background about Citect server/client operations may be helpful. As clients start up, they first try the server as defined in [Client]Primary= (Citect.ini file). Failing that, they then attempt to connect to [Client]Standby=. Once a connection is made, it is not broken until a failure or a ClusterSetName command. Even if a client connects to the standby, it will not try to connect to the primary until the standby fails or goes offline. Armed with this knowledge, some users and system designers configure half their clients to connect to the primary server (call it Citect1) first, then to the standby (Citect2) failing that. The other half of the clients attempt to connect to Citect2 first, then Citect1. This is an attempt to loadshare, but it only works until one of the servers goes offline for whatever reason. At this point, dependent clients switch to the remaining server and remain there, even after their "primary" server once again becomes available.

Incidentally, there is no limit to the number of Citect servers that can be operational on a given Citect network. Designers can plan to have any number of alarm servers for instance. However, clients can only connect to one server at a time, with the other name in the cluster pair held ready for immediate connection if required.

## **Alarms**

Alarm servers are not truly hot standby, neither are they redundant. They are hot standby in the sense that only one of them is responsible for printing out alarm summaries etc and executing alarm-based code. The “On Action” and “Off Action” events (defined in Alarm Categories) are examples of this kind of code. The “Primary” alarm server is responsible for the processing of these tasks. If the primary fails, the standby alarm server can take up this processing load. The primary server is defined as such by the [Alarm]Primary= parameter in the Citect.ini file. They are redundant in that they each scan all the alarms all the time. Each server catches up on current alarm states on startup (from the other operational server), and then essentially strikes out on its own. This arrangement normally works well since both servers share the same data source, and so all alarms raised by one server will be accordingly raised by the other.

Depending on alarm activity clients are sent a stream of alarm data by the server upon connection. Normally certain alarm traffic is actually pushed to clients, which in this case (alarms) is more efficient. When a client attempts to call up an alarm page – the client then makes a request for all alarm data in order to complete the display. Alarm info can be sent in clear text or by alarm record identifiers. In fact, the client does not need to have the alarms configured locally at all if clear text transmission is used. Once the client connects to the server, it receives all the messages as is.

When a client acknowledges an alarm message, this is recorded on the server the client is currently connected to and then a subsequent message is sent from that server to the other to acknowledge the alarm there as well. This is the only form of ongoing synchronisation between alarm servers. Therefore it is possible for a pair of redundant alarm servers to have slightly different alarm lists, since each server is asking for data which over time is going to change. Alarms that don't latch (extremely transient and fleeting) as well as request cycles which are slightly off from a timing point of view could exacerbate this situation but in most cases this problem simply won't occur.

## **Trends**

Trend servers are redundant. Each trend server is fully operational at all times, sampling data and archiving the results. There is no real concept of a primary or standby trend server, since they each do the same task. When a client wants to display a trend it requests the data according to trend tag. If the operator knows the trend tag then it is technically possible to leave all trend definitions out of the client project and allow them typed them in on the fly, since the tag is all that's required to extract the data from the server. Initially, a large chunk of data is requested by the client (to draw the entire display). Subsequently, small updates are requested to keep the trend current. This client request rate is defined by the [Trend]ClientRequestTime parameter.

Citect trend servers have built in functionality to allow them to backfill an interval of missing data that occurs during a shutdown or failure. This is trend server

redundancy. When a trend server starts up, it notes all the samples that are missing (between when it was last running and “now”) and submits a request to the redundant server for the missing data. This data is sent across, the trend files are updated and the server recommences trending operations. This chain of events can take some minutes to complete, depending on the number of trends, the length of time the server was down and the sample rate of the trends. If the redundant server can not be contacted for the purposes of backfilling a gap in trend data then the server immediately begins trending again and the gap remains permanently. The same situation can arise when the second trend server is shutdown before the backfill operation has completed. The easiest way to fix a hole in a trend is to copy files from the good server over the top of the bad files on the starting server. Be advised however that in many cases what is missing on one server will be present on the other, and vice versa. It should be noted that it is possible to write cicode to patch up trend files, one from another. This may be the only option in complex redundant server operations.

This “update on startup” behaviour gives rise to another situation which is not so desirable. If the network connection between a trend server and the IO server fails, then the trend server will fill the trend file with “n/a” samples. This does not constitute a hole, and does not get filled on restart. The best way around this is to simply shut down the trend server, and restart it when the IO server is up again. Alternatively, the aforementioned potential cicode solution (if one exists) could get the system out of trouble here.

## Reports

Report servers complete the picture in that they are hot standby. During normal operation, the primary report server completes all tasks, while the standby machine waits for a failure or shutdown. Upon the failure/shutdown of the primary, the standby assumes responsibility for report tasks and begins actively searching for the primary using a network poll. When the primary responds to this poll, it is assumed to be once more operational and report tasks cease operation on the standby, their processing be taken over by the primary again.

## **Disclaimer**

© 2006 Citect, Inc. All rights reserved. The information contained in this document represents the current view of Citect on the issues discussed as of the date of publication. Because Citect must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Citect, and Citect cannot guarantee the accuracy of any information presented after the date of publication. This white paper is for informational purposes only. CITECT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in, or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording or otherwise) or for any purpose, without the express written permission of Citect, Inc. Citect may have patents, patent applications, trademarks, copyrights or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Citect, the furnishing of this document does not give you any license to these patents, trademarks, copyrights or other intellectual property. Citect, CitectSCADA, CitectHMI, CitectSCADA Reports, Ampla and Meta are either registered trademarks or trademarks of Citect Group Corporation in Australia and/or other countries. The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

