Caché Distributed Data Management Guide

Version 2018.1
2024-11-07

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Caché Distributed Data Management Guide
PDF generated on 2024-11-07
InterSystems Caché® Version 2018.1

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

List of Tables

About This Book

One of the most powerful and unique features of Caché is the ability to efficiently distribute data and application logic among a number of server systems using the Enterprise Cache Protocol (ECP). This guide contains chapters on configuring and managing databases with ECP, and an appendix of guarantees and limitations:

- ECP Overview
- Configuring Distributed Systems
- Monitoring Distributed Applications
- Developing Distributed Applications
- ECP Recovery Guarantees and Limitations

For detailed information, see the Table of Contents.

For general information, see *Using InterSystems Documentation*.

1

ECP

One of the most powerful and unique features of Caché is the ability to efficiently distribute data and application logic among a number of server systems.

The underlying technology behind this feature is the Enterprise Cache Protocol (ECP): a distributed data caching architecture that manages the distribution of data and locks among a heterogeneous network of server systems.

Unlike other "multi-tier" architectures, ECP is primarily a configuration option. That is, you do not have to use special code or development techniques to create distributed database applications. This provides several important advantages over other technologies:

- Applications can scale *down* as well as up. An application that is truly scalable can run on small systems as well as large using the same code base. With Caché you can deploy small scale systems using a single server and deploy the same application at large sites on multiple servers using ECP.
- Applications are *easier* to develop. Instead of worrying about scalability and infrastructure, application developers can focus on core, customer-centric functionality.
- Applications are *reliable*. ECP automatically recovers from most runtime problems (such as planned or unplanned system stoppages) with no user intervention.

This overview covers the following topics:

1.1 ECP Features

ECP provides the following features:

- *Automatic operation.* Once configured, ECP automatically establishes and maintains connections between application servers and data servers.
- *Fail-safe operation.* ECP automatically attempts to recover from any disconnections (planned or unplanned) among application server and data server systems. When it reestablishes a broken connection, ECP automatically restores the operating state of the system: it resumes all open transactions, restores locks, and recovers all database changes made by the application server.

If it cannot recover a connection in a reasonable time, ECP automatically rolls back any outstanding transactions involving the connection.

- *Heterogeneous networking.* Caché systems in an ECP configuration can run on different hardware and operating system platforms. ECP automatically manages any required data format conversions.
- *A shared network buffer cache.* ECP uses a portion of the Caché general database buffer pool to cache data retrieved across the network. This cache is shared among all Caché processes on an ECP application server system.
- *A robust transport layer based on TCP/IP.* ECP uses the standard TCP/IP protocol for data transport, making it easy to configure and maintain.
- *Efficient use of network bandwidth.* ECP is designed to take full advantage of the latest-generation, high-performance, networking infrastructures.

Besides providing a high degree of system availability, the automatic behavior of ECP makes a system easier to manage. For example, it is possible to take an ECP data server offline temporarily for a software upgrade and restore it without having to perform any operations on the ECP application server systems.

1.2 Uses for ECP

The primary reasons to use ECP are:

- To provide *greater scalability* for applications, especially applications that are computationally-bound (that is, they are limited by the number of available CPU cycles and not by I/O operations).
- As part of an application *failover strategy* for high availability systems. For more information see the "System Failover Strategies" and "ECP Failover" chapters of the *Caché High Availability Guide*.

For information on configuring a system for ECP, see the "Configuring Distributed Systems" chapter.

1.3 ECP Architecture

The architecture and operation of ECP is conceptually simple. ECP provides a way to efficiently share data, locks, and executable code among multiple Caché systems. Data and code are stored remotely, but are cached locally to provide efficient access with minimal network traffic.

For more information on application development and design using ECP, see the "Developing Distributed Applications" chapter.

1.3.1 Databases and Namespaces

To better understand how ECP works, it is first helpful to review how databases, namespaces, and caching work in Caché.

Caché stores data—persistent multidimensional arrays (globals) as well as executable code (routines)—in one or more physical structures called *databases*. A database consists of one or more physical files stored in the local operating system. A Caché system may (and usually does) have multiple databases.

Each Caché system maintains a *database cache*—a local, shared memory buffer used to cache data retrieved from the physical databases. This cache greatly reduces the amount of costly I/O operations required to access data and provides many of the performance benefits of Caché.

Caché applications access data by means of a *namespace*. A namespace provides a logical view of data (globals and routines) stored in one or more physical databases. A Caché system may (and usually does) have multiple namespaces.

Caché maps the data visible in a *logical* namespace to one or more *physical* databases. This mapping provides applications with a powerful mechanism for changing an application's physical deployment (which disk drives are used, etc.) without changing application logic. This same mechanism, in conjunction with ECP, is what makes it possible to redeploy applications among multiple, networked systems with few or no application changes.

1.3.2 ECP Application Servers and Data Servers

An ECP configuration consists of a number of Caché systems that are visible to one another across a TCP/IP-based network. There are two roles a Caché system can play in an ECP configuration:

- *ECP Data Server* — a Caché system that is *providing* data for one or more ECP application server systems.
- *ECP Application Server* — a Caché system that is *consuming* data provided by one or more ECP data server systems.

A Caché system can simultaneously act as both an ECP data server and an ECP application server. However, one Caché instance cannot act as an ECP data server for the data it receives as an application server of another ECP data server.

In an ECP configuration, each *ECP data server* is responsible for the following:

- Storing data in its local database.
- Maintaining the coherency of the various ECP application server system database caches so that application servers do not see stale data.
- Managing the distribution of locks across the network.

In an ECP configuration, each *ECP application server* is responsible for the following:

- Establishing connections to a specific ECP data server whenever an application requests data that is stored on that server.
- Monitoring the status of all connections to ECP data servers. If a connection is broken, or encounters any trouble, the ECP application server attempts to recover the connection.
- Maintaining, in its cache, data retrieved across the network. This cache greatly reduces the number of costly network operations needed to access remote data.

1.3.3 ECP Connections and Recovery

ECP automatically establishes and maintains network connections between application server and data server systems. If a connection is broken, ECP automatically reestablishes the connection and restores the operating state of the system, if possible.

For more information on ECP connections, see the "Monitoring Distributed Applications" chapter.

For more information on ECP recovery, see the "Developing Distributed Applications" chapter.

2

Configuring Distributed Systems

An ECP application consists of one or more ECP data server systems — data providers — distributing to one or more ECP application server systems — data consumers. The primary means of configuring an ECP application is using the ECP Settings page of the Management Portal (**System Administration** > **Configuration** > **Connectivity** > **ECP Settings**).

Once you have decided how to distribute your data, configuring an ECP application is very straightforward:

1. Enable each system that provides data as an ECP data server. See the Configuring an ECP Data Server section for instructions.
2. If you are using Security, see the Managing ECP Privileges section for a discussion on how resources, roles, and privileges are managed in an ECP configuration.
3. Specify each system that requests data as an ECP application server for each data server with which it wishes to communicate. See the Configuring an ECP Application Server section for instructions.
4. In addition, configure each ECP application server system so that it can see remote data in the defined ECP data servers. See the Configuring ECP Remote Data Access section for instructions.
5. ECP shares the buffer pool with the local instance of Caché; therefore, InterSystems recommends allocating additional buffers to accommodate ECP. See the Memory Use on Large ECP Systems section of the "Developing Distributed Applications" chapter of this guide for details.

A system operating as an ECP data server can simultaneously act as an ECP application server, and vice versa. You may configure your ECP application and data servers in any order; you do not need to enable an ECP data server before defining an application server.

2.1 Configuring an ECP Data Server

To configure a system as an ECP data server, you must first enable the ECP service from the Services page of the Management Portal (**System Administration** > **Security** > **Services**). Click **%Service_ECP**, select the **Service enabled** check box, and click **Save**. This is the only configuration setting required to use this system as an ECP data server.

Alternatively, from the ECP Settings page, click **Edit** next to **The ECP service is Disabled** to navigate to the same **Edit Service** page. When you click **Save**, you return to the **ECP Settings** page.

To see a list of ECP application servers that have been configured to connect to this data server, click the **Application Servers** button on the ECP Settings page.

Note: For a detailed explanation of Caché services, see the "Services" chapter of the *Caché Security Administration Guide*.

Update the **Maximum number of application servers** setting to specify the maximum number of application servers that can possibly access this data server simultaneously. Caché allocates a limited number of application server nodes. Increase the default value of `1` up to a maximum of `254` to avoid a system restart, which is required when the number of connections becomes greater than the number of allocated nodes.

Note: If you increase the maximum number of application server, you must restart Caché.

The ECP data server is now ready to accept connections from valid ECP application servers.

You may wish to restrict access to the data server. See the following sections for ways to do this:

- Restricting ECP Application Server Access
- Specifying ECP Privileges and Roles

2.1.1 Restricting ECP Application Server Access

You can restrict which systems can act as ECP application servers for an ECP data server system by performing the following steps:

1. From the Services page, click **%Service_ECP**.
2. In the **Allowed Incoming Connections** box, click **Add** and enter a single address (for example, `192.9.202.55` or `mycomputer.myorg.com`) or a range of addresses (for example, `18.61.202–210.*` or `18.68.*.*`).

If you enter IP addresses in the **Allowed Incoming Connections** list, the ECP data server only accepts incoming ECP connections from application servers whose IP is in the list. If the list is empty, any application server can connect to this system if the ECP service is enabled.

After you add an IP address, it appears in the list of **Allowed Incoming Connections** with options to **Delete** the address from the list and **Edit** the **Roles** of the connection.

This process of managing roles on ECP data and application servers is part of Caché security. For details on how Caché roles and privileges work in general see the "Roles" chapter of the *Caché Security Administration Guide*. The following section details how these features work with ECP.

2.1.2 Specifying ECP Privileges and Roles

For each specified IP address or range of addresses, click **Edit** to display the **Select Roles** area that allows you to specify the roles associated with the connection from the IP address. By default, the connection holds the %All role. If you specify one or more other roles, these roles are the only roles that the connection holds. Hence, a connection from an IP address with the %Operator role has only the privileges associated with that role, while a connection from a different IP address with no associated roles (and therefore %All) has all privileges.

To specify the roles associated with an IP address:

1. Select roles from those listed under **Available** and click the right arrow to add them to the **Selected** list.
2. To remove roles from the **Selected** list, click them and then click the left arrow.
3. To add all roles to the **Selected** list, click the double right arrow; to remove all roles from the **Selected** list, click the double left arrow.
4. Click **Save** to associate the roles with the IP address.

The Managing ECP Privileges section discusses how Caché manages privileges within an ECP configuration.

2.2 Managing ECP Privileges

The following discussion assumes that resources and roles refer to the same assets on each machine. To be granted access to a resource on the ECP data server, the role held by the process on the application server and the role set for the ECP connection on the data server must both include permissions for the same resource.

By default, Caché grants the ECP data server the %All privilege when the data server runs on behalf of an ECP application server. This allows it to return any data in any database that the application server requests. Caché restricts access to this data on the application server based on the privileges of the user requesting the data on the application server.

For example, for a user on the application server who only has privileges for the **%DB_USER** resource, data in the USER database on the data server is available (which by default is assigned the **%DB_USER** resource), but attempting to access the SAMPLES database on the data server results in a <PROTECT> error. If a different user on the application server has privileges for the **%DB_SAMPLES** resource, then the SAMPLES database on the data server is available.

You can also restrict the set of roles on the data server based on the IP Address of the application server. For example, on the data server you can specify that when interacting with an application server named NODE_A the only available role is **%DB_USER**. In this case, users on the application server granted the **%DB_USER** role can access the USER database on the data server. However, users on the application server with **%DB_SAMPLES** access receive a <PROTECT> error if they try to access the SAMPLES database on the data server (since the data server is only set up with **%DB_USER** access).

The following are exceptions to this behavior:

- Caché always grants the ECP data server the %DB_CACHESYS role since it requires Read access to the CACHESYS database to run. This means that a user on an ECP application server with %DB_CACHESYS can access the CACHESYS database on the ECP data server.

  To prevent a user on the application server from having access to the CACHESYS database on the data server, there are two options:

  – Do not grant the user privileges for the %DB_CACHESYS resource.

  – On the data server, change the name of the resource for the CACHESYS database to something other than %DB_CACHESYS, making sure that the user on the application server has no privileges for that resource.

- If the ECP data server has any public resources, they are available to any user on the ECP application server, regardless of either the roles held on the application server or the roles configured for the ECP connection.

Changes both to the configuration of the ECP connection and to the public permissions on resources require a restart of Caché before taking effect.

Security-related ECP Error Reporting

The behavior of security-related error reporting with ECP varies depending on whether the check fails on the application server or the data server and the type of operation:

- If the check fails on the application server, there is an immediate <PROTECT> error.
- For synchronous operations on the data server, there is an immediate <PROTECT> error.
- For asynchronous operations on the data server, there is a possibly delayed <NETWORK DATA UPDATE FAILED> error. This includes **Set** operations.

2.3 Configuring an ECP Application Server

To configure a system as an ECP application server, you define an ECP data server from which to retrieve data. Add this remote ECP data server by performing the following steps:

1. From the ECP Settings page, click **Data Servers** to display a list of currently configured ECP data servers.
2. Click **Add Server** to add a data server.
3. Enter the following information for the data server:
   - **Server Name** — Enter a logical name for the convenience of the application system administrator.
   - **Host DNS Name or IP Address** — Specify the host name either as a raw IP address (in dotted-decimal format or, if IPv6 is enabled, in colon-separated format) or as the Domain Name System (DNS) name of the remote host. If you use the DNS name, it resolves to an actual IP address each time the application server initiates a connection to that ECP data server host. For more information, see the IPv6 Support section in the "Configuring Caché" chapter of the *Caché System Administration Guide*.

     Important: When adding a mirror as an ECP data server, do not enter the virtual IP address (VIP) of the mirror, but rather the DNS name or IP address of the current primary failover member. Because the application server regularly collects updated information about the mirror from the specified host, it automatically detects a failover and switches to the new primary failover member. See the "Mirroring" chapter of the *Caché High Availability Guide* for information about mirror failover and VIPs.

   - **IP Port** — The port number defaults to `1972`; change it as necessary to the superserver port of the Caché instance on the data server.
   - Select the **Mirror Connection** check box if this data server is the primary failover member of a mirror.
4. Click **Save**.

Once you add a remote ECP data server, it appears in the list of defined data servers this application server can connect to at the bottom of this same portal page. Add additional ECP data servers to the list using the **Add Remote Data Server** link. Remove or edit server definitions using the **Delete** and **Edit** links, respectively. You may also click **Change Status** of the connection. See the "Monitoring Distributed Applications" chapter for details.

You may add as many data servers as allowed by the **Maximum number of data servers** setting. Update this value to specify the maximum number of server connections the application server may need later so that Caché reserves enough system resources so as not to require a restart each time you add a data server. Increase the default value of `2` up to a maximum of `254`.

Note: If you increase the maximum number of data servers, you must restart your Caché.

Your system is ready to act as an ECP application server. No further user intervention is required; when the ECP application server needs access to the ECP data server, it automatically establishes a connection to the server.

2.4 Configuring ECP Remote Data Access

After defining a list of one or more ECP data servers for an ECP application server, configure the ECP application server system so that it has access to data stored in the ECP data server system. Do this by defining a remote database on the ECP application server system.

A *remote database* is a database that is physically located on an ECP data server system, as opposed to a *local database* which is physically located on the local application server system.

To define a remote database on the ECP application server, perform the following steps:

1. Navigate to the Remote Databases page of the Management Portal (**System Administration** > **Configuration** > **System Configuration** > **Remote Databases**).
2. Click **Create New Remote Database** to invoke the **Database Wizard**, which displays a list of the logical names (the name you used when you added it to the list of ECP data servers) of the remote data servers on the application server.
3. Click the name of the appropriate ECP data server and click **Next**.
4. The portal displays a list of database directories on the remote ECP data server. Select one of these to serve as the remote database.
5. Enter a database name (its name on the ECP application server; it does not need to match its name on the ECP data server) and click **Finish**. You have defined a remote database.

Next, define a new namespace (or modify an existing namespace) to view the data in the remote database as you would in a local database.

Note: By using the **Namespace Wizard** in the Management Portal, you can define a namespace and a remote database at the same time, thereby combining these two procedures for adding a remote database.

To define a new namespace that views the data in a remote database perform the following steps:

1. Navigate to the Namespaces page of the Management Portal (**System Administration** > **Configuration** > **System Configuration** > **Namespaces**).
2. Click **Create New Namespace**.
3. Fill in the form with the following fields:
   - Enter a name for the new namespace.
   - Click **Remote Database**.
   - If you created a remote database as described previously, select it; otherwise click **Create New Database** and follow the previous Database Wizard instructions.
   - If you use CSP, select **Create a default CSP application for this namespace**.
4. Choose a database for the new namespace. Select the remote database from the list (remote and local databases are listed together) and click **Next**.
5. Click **Save**. You have a new namespace that is mapped to a remote database.

Any data retrieved or stored in this namespace is loaded from and stored in the physical database on the ECP data server and updated in the local application server system cache if it is already cached.

2.5 ECP Security Notes

First, all the instances in an ECP configuration need to be within the secured Caché perimeter (that is, within an externally secured environment). This is because:

- ECP is a basic security service (not a resource-based service), so there is no way to regulate which users have access to it. For more information on basic and resource-based services, see the "Available Services" section of the "Services" chapter of the *Caché Security Administration Guide*.
- Caché does not support SSL/TLS to secure ECP connections. For more information on the use of SSL/TLS, see the "Using SSL/TLS with Caché" chapter of the *Caché Security Administration Guide*.

Also, when using encrypted databases on ECP data servers, it is recommended to encrypt the CACHETEMP database on all connected application servers. The same or different keys can be used. For more information on database encryption, see the "Managed Key Encryption" chapter of the *Caché Security Administration Guide*.

3

Monitoring Distributed Applications

A running ECP application consists of one or more ECP data server systems—data providers—connected to one or more ECP application server systems—data consumers. Between each application server and data server that share data, there is an *ECP connection*: a TCP/IP connection that ECP uses to send data and commands.

You can monitor the status of the servers and connections in an ECP application from the ECP Settings page of the Management Portal (**System Administration** > **Configuration** > **Connectivity** > **ECP Settings**).

The ECP Settings page has two subsections:

1. **This System as an ECP Data Server** displays settings for the data server as well as the status of the ECP service. See "Configuring Distributed Systems" for more information.
2. **This System as an ECP Application Server** displays settings for the application server and a list of data servers—systems providing data *to* this node—connected to this application server as well as their status.

The following sections describe status information for connections:

- ECP Connection Information
- ECP Connection States
- ECP Connection Operations

3.1 ECP Connection Information

The ECP Settings page displays a list of the current ECP data server connections on the application server side. The ECP Application Servers page (click **Application Servers** on the ECP Setting page) displays a list of the current ECP application server connections on the data server side of a connection.

3.1.1 ECP Data Server Connections

The **This System as an ECP Application Server** list displays the following information for each *ECP data server* connection:

Server Name

The logical name of the ECP data server system on the application server for this connection.

Host Name

The host name of the ECP server system for this connection as entered when the server was added to the application server configuration.

IP Port

The IP port number used to connect to the ECP server system.

Status

The current status of this connection. Each connection has a current operating state. These states are described in the ECP Connection States section.

Edit

If the current status of this connection is not connected or disabled, you can edit the port and host information of the data server.

Change Status

From each data server row you can change the status of an existing ECP connection with that data server. See the ECP Connection Operations section for more information.

Delete

You can delete the data server information from the application server side.

3.1.2 ECP Application Server Connections

Click **ECP Application Servers** to view a list of ECP application servers that are connected to this system:

The **This System as an ECP Application Server** list displays the following information for each *ECP data server* connection:

Client Name

The logical name of the ECP application server system on the data server for this connection.

Status

The current status of this connection. Each connection has a current operating state. These states are described in the ECP Connection States section.

Client IP

The host name of the ECP server system for this connection as entered when the server was added to the data server configuration.

IP Port

The IP port number used to connect to the ECP server system.

3.2 ECP Connection States

In a running system, an ECP connection can be in one of the following states:

Table 3–1: ECP Connection States

State	Description
Not Connected	The connection is defined but has not been used yet.
Connection in Progress	The connection is in the process of establishing itself. This is a transitional state that lasts only until the connection is established.
Normal	The connection is operating normally and has been used recently.
Trouble	The connection has encountered a problem. If possible, the connection automatically corrects itself.
Disabled	The connection has been manually disabled by a system administrator. Any application making use of this connection receives a `<NETWORK>` error.

The following sections describe each connection state as it relates to being on the application server or data server side:

- Application Server Connection States
- Data Server Connection States

3.2.1 Application Server Connection States

The following sections describe the application server side of each of the connection states:

- Not Connected
- Connection in Progress
- Normal
- Trouble
- Transitional Recovery
- Disabled

Application Server Not Connected State

An application server-side ECP connection starts out in the *Not Connected* state. In this state, there are no ECP daemons for the connection. If an application server process makes a network request, daemons are created for the connection and the connection enters the *Connection in Progress* state.

Application Server Connection in Progress State

In the *Connection in Progress* state, a network daemon exists for the connection and actively tries to establish a new connection. A user process must wait until the connection completes before it can submit requests to the network. While the connection is in the *Connection in Progress* state, the user process waits on each request for up to 20 seconds for the connection to complete. When the connection is established, it enters the *Normal* state. If the connection is not established within that time, the user process receives a `<NETWORK>` error.

The application server ECP daemon attempts to create a new connection to the data server in the background. If no connection is established within 20 minutes, the connection returns to the *Not Connected* state and the daemon for the connection goes away.

Application Server Normal State

After a connection completes, it enters the *Normal*, data transfer, state. In this state, the ECP application server-side daemons exist and actively send requests and receive answers across the network. The connection stays in the *Normal* state until the connection becomes unworkable or until the application server or the data server requests a shutdown of the connection.

Application Server Trouble State

If the connection from application server to data server encounters problems, the application server ECP connection enters the *Trouble* state. In this state, application server ECP daemons exist and actively try to restore the connection. An underlying TCP connection may or may not still exist. The recovery method is similar whether or not the underlying TCP connection gets reset and must be recreated, or if it stops working temporarily.

During the application server *Trouble* state interval, the application server attempts to reconnect to the data server to perform ECP connection recovery. During this interval, existing network requests are preserved. The originating application server-side user process blocks new network requests, waiting for the connection to resume. If the connection returns within the trouble timeout (**Time to wait for recovery** currently defaults to 20 minutes), it returns to the *Normal* state and the blocked network requests proceed.

For example, if a data server goes offline, any application server connected to it has its state set to *Trouble* until the data server becomes available. If the problem is corrected gracefully, a connection's state reverts to *Normal*; otherwise, if the trouble state is not recovered, it reverts to *Not Connected*.

Applications continue running until they require network access. All locally cached data is available to the application while the server is not responding.

Application Server Transitional Recovery States

Transitional recovery states are part of the *Trouble* state. If there is no current TCP connection to the data server, and a new connection is established, the application server and data server engage in a *recovery protocol* which flushes the application server cache, recovers transactions and locks, and returns to the *Normal* state.

Similarly, if the data server shuts down, either gracefully or as a result of a crash, and then restarts, it enters a short period (approximately 30 seconds) where it allows application servers to reconnect and recover their existing sessions. Once again, the application server and the data server engage in the recovery protocol.

If connection recovery is not complete within 20 minutes, the application server gives up on connection recovery. Specifically, the application server returns errors to all pending network requests and changes the connection state to *Not Connected*. If it has not already done so, the data server rolls back all the transactions from this application server and releases all the locks from this application server the next time this application server connects to the data server.

If the recovery is successful, the connection returns to the *Normal* state and the blocked network requests proceed.

Application Server Disabled State

An ECP connection is marked *Disabled* if an administrator declares that it is disabled. In this state, no daemons exist and any network requests that would use that connection immediately receive `<NETWORK>` errors.

3.2.2 Data Server Connection States

The following sections describe the data server side of each of the connection states:

- Free
- Normal
- Trouble

- Recovering

Data Server Free States

When an ECP server first comes up, all incoming ECP connections are in an initial "unassigned" *Free* state and are available for connections from any ECP application server that is listed in the connection access control list. If a connection from an application server previously existed and has since gone away, but does not require any recovery steps, the connection is placed in the "idle" *Free* state. The only difference between these two states is that in the idle state, this connection block is already assigned to a particular application server, rather than being available for any application server that passes the access control list.

Data Server Normal State

In the data server *Normal* state, the application server connection is normal. At any point in the processing of incoming connections, whenever the application server disconnects from the data server (except as part of the data server's own shutdown sequence), the data server rolls back any pending transactions and releases any incoming locks from that application server, and places the application server connection in the "idle" *Free* state.

Data Server Trouble States

If the application server is not responding, the data server shows a *Trouble* state. If the data server crashes or shuts down, it remembers the connections that were active at the time of the crash or shutdown. After restarting, the data server waits for a brief time (usually 30 seconds) for application servers to reclaim their sessions (locks and open transactions). If an ECP application server does not complete recovery during this awaiting recovery interval, all pending work on that connection is rolled back and the connection is placed in the "idle" state.

Data Server Recovering State

The data server connection is in a recovery state for a very short time when the application server is in the process of reclaiming its session. The data server keeps the application server in trouble state for a brief time (**Time interval for Troubled state** currently defaults to 60 seconds) for it to reclaim the connection; otherwise, it releases the application resources (rolls back all open transactions and releases locks) and then sets the state to *Free*.

3.3 ECP Connection Operations

From the ECP Settings page of the Management Portal on an ECP application server, you can change the status of the ECP connection. From each data server row, click **Change Status** to display the connection information and perform the appropriate choices of the following:

Change to Disabled

Set the state of this connection to *Disabled*. This releases any locks held for the ECP application server, rolls back any open transactions involving this connection, and purges cached blocks from the data server. If this is an active connection, the change in status sends an error to all applications waiting for network replies from the data server.

Change to Normal

Set the state of this connection to *Normal*.

Change to Not Connected

Set the state of this connection to *Not Connected*. As with changing the state to disabled, this releases any locks held for the ECP application server, rolls back any open transactions involving this connection, and purges cached blocks from the data server. If this is an active connection, the change in status sends an error to all applications waiting for network replies from the data server.

4

Developing Distributed Applications

This chapter discusses application development and design issues that are helpful if you would like to deploy your application using ECP, either as an option or as its primary configuration.

With Caché, the decision to deploy an application as a distributed (multiserver) system is primarily a runtime configuration issue (see Configuring Distributed Systems). Using the Caché configuration tools, map the logical names of your data (globals) and application logic (routines) to physical storage on the appropriate system.

This chapter discusses the following topics:

- ECP Recovery
- Forced Disconnects
- Performance and Programming Considerations
- ECP-related Errors

4.1 ECP Recovery

ECP is designed to automatically recover from interruptions in connectivity between the ECP application server and the ECP data server.

If the connection between an ECP application server and ECP data server is interrupted, the following happens:

1. The application server connection state is set to *Trouble* indicating that this connection is attempting to recover.
2. The ECP application server attempts to reestablish its connection with the ECP data server.
3. If the application server connection is reestablished within the **Time interval for Troubled state** (see ECP Timeout Values), the connection state changes to *Normal*. ECP restores all locks and open transactions to the state they were in prior to the interruption.
4. If the ECP application server cannot reestablish its connection with the ECP data server within the **Time interval for Troubled state**, it disables the connection, returns a `<NETWORK>` error to all processes waiting for remote activities, and then sets the application server connection state to *Not Connected*. The next reference establishes a new ECP connection to the data server.
5. If the connection is not reestablished within the **Time to wait for recovery** (see ECP Timeout Values), the ECP data server rolls back any open transactions involving the ECP application server and then releases all locks held on the server on behalf of the ECP application server. The state of the data server connection changes to *Free*.

By default, ECP uses the following timeout values:

Table 4–1: ECP Timeout Values

Management Portal Setting	Default Value	Range
Time interval for Troubled state	60 seconds	20–65535 seconds
Time to wait for recovery	1200 seconds (20 minutes)	10–65535 seconds
Time between reconnections	5 seconds	1–60 seconds

Each is configurable from the ECP Settings page of the Management Portal (**System Administration** > **Configuration** > **Connectivity** > **ECP Settings**). They are described in the following:

Time Interval for Troubled State

The duration a connection stays in troubled state (in seconds). Once this period of time has elapsed, the data server declares the connection dead and presumes recovery is not possible.

The setting is in the **This System as an ECP Data Server** section; the default value is 60 seconds. The range is 20–65535 seconds.

Time to wait for recovery

How long an application server should keep trying to reestablish a connection before giving up or declaring the connection failed (in seconds).

The setting is in the **This System as an ECP Application Server** section; the default value is 1200 seconds (20 minutes). The value should be at least 10 seconds and can be a maximum of 65535 seconds. The application server continues reconnection attempts as scheduled by the reconnect interval until the full duration expires.

Time between reconnections

How long to wait between each reconnection attempt (in seconds), when a data server is not available.

The setting is in the **This System as an ECP Application Server** section; the default value is 5 seconds. The range is 1–60 seconds. The application server continues reconnection attempts at intervals scheduled by this interval until the full reconnect duration expires.

The default values are set so the data server gives up quicker because Caché does not want to tie up data server resources for an extended amount of time for an application server that is down. The application server waits for up to twenty minutes because when data servers crash and restart, Caché wants to give the application server a chance to complete recovery after the data server comes back up. There is an implicit assumption that a data server has something better to do with its time than wait for an application server to reconnect, but an application server does not.

ECP relies on the TCP physical connection to detect the health of the other side of the pipe. ECP ensures there is always a message going across without flooding the pipe. On most platforms you can adjust the TCP connection failure and detection behavior at the system level.

If no traffic is received from an application server for a while, the data server declares the connection non-responsive. In the non-responsive state, there is an active ECP data server daemon that is waiting for new requests to arrive on the connection, or for a new connection to be requested. If the old connection returns, it can immediately resume operation without recovery. Because of the underlying heartbeat mechanism, if the application server goes away completely, either because of application server failure or network failure, the underlying TCP connection is quickly reset. Thus, a long time in the non-responsive state on the data server generally indicates some kind of problem on the application server (a system hang, for example) that caused the application server to stop functioning, but without interfering with its connections.

If the underlying TCP connection is reset, the data server puts the connection in an "awaiting reconnection" state. In the "awaiting reconnection" state, there is no active ECP daemon on the data server. A new pair of ECP data server daemons will be created when the next incoming connection is requested by the application server system.

Collectively, the non-responsive state and the awaiting-recovery state are known as the data server-side *Trouble* state. The recovery required in both cases is very similar.

If the data server crashes or shuts down, it remembers the connections that were active at the time of the crash or shutdown. After restarting, the data server has a short window (usually 30 seconds) during which it places these interrupted connections in an "awaiting reconnection" state. In this state, the application server and data server can cooperate together to recover all the transaction and lock states as well as all the pending **Set** and **Kill** transactions from the moment of the data server shutdown.

During the recovery of an ECP-configured system, Caché guarantees a number of recoverable semantics which are described in detail in the ECP Recovery Guarantees section of the "ECP Recovery Guarantees and Limitations" appendix. There are limitations to these guarantees which are described in detail in the ECP Recovery Limitations section of the same appendix.

4.2 Forced Disconnects

By default, ECP automatically manages the connection between an application server system and a data server system. When an ECP-configured system is initially started, all connections between ECP application servers and data servers are in the *Not Connected* state (that is, the connection is defined, but not yet established). As soon as an ECP application server makes a request (for data or code) that requires a connection to an ECP data server, the connection is automatically established and the state changes to *Normal*. The network connection between the ECP application server and data server is kept open indefinitely.

In some applications, you may wish to close open ECP connections. For example, suppose you have a system, configured as an ECP application server, that periodically (a few times a day) needs to fetch data stored on a data server system, but does not need to keep the network connection with the data server open afterwards. In this case, the ECP application server system can issue a call to the **ChangeToNotConnected** method of the SYS.ECP class to force the state of the ECP connection to *Not Connected*.

For example:

ObjectScript

```
Do OperationThatUsesECP()
Do SYS.ECP.ChangeToNotConnected("ConnectionName")
```

The **ChangeToNotConnected** method does the following:

1. Completes sending any data modifications to the data server and waits for acknowledgment from the data server.
2. Removes any locks on the ECP data server that were opened by the ECP application server.
3. Rolls back the data server side of any open transactions. The application server side of the transaction goes into a "rollback only" condition.
4. Completes pending requests with a <NETWORK> error.
5. Flushes all cached blocks.

After completion of the state change to *Not Connected*, the next request for data from the ECP data server automatically reestablishes the connection.

4.3 Performance and Programming Considerations

To achieve the highest performance and reliability from ECP applications, you should be aware of the following issues:

4.3.1 Avoid Transactions Spanning Multiple Data Servers

Restrict updates within a single transaction to a single server, whether it be a remote ECP data server or the local server. When a transaction includes updates to more than one server (including the local server) and the **TCommit** cannot complete successfully, some servers that are part of the transaction may have committed the updates while others may have rolled them back. See the *Commit Guarantee* section of "ECP Recovery Guarantees and Limitations" for details.

Note: Updates to CACHETEMP are not considered part of the transaction for the purpose of rollback, and, as such, are not included in the restriction.

4.3.2 ZSync Command Nonfunctional

In ECP configurations, the **ZSync** command is not operable. ECP achieves an analogous effect within transactions by declaring the entire transaction as *rollback-only* whenever any **Set** or **Kill** operation receives an error.

4.3.3 Memory Use on Large ECP Systems

In addition to buffering the blocks that are served over ECP, ECP data servers use global buffers to store various ECP control structures. There are several factors that go into determining how much memory these structures might require, but the most significant is a function of the aggregate sizes of the clients' caches. To roughly approximate the requirements, so you can adjust the data server's database caches if needed, use the following guidelines:

Database Block Size	Recommendation
8 KB	50 MB plus 1% of the sum of the sizes of all of the application servers' 8 KB database caches
16 KB (if enabled)	0.5% of the sum of the sizes of all of the application servers' 16 KB database caches
32 KB (if enabled)	0.25% of the sum of the sizes of all of the application servers' 32 KB database caches
64 KB (if enabled)	0.125% of the sum of the sizes of all of the application servers' 64 KB database caches

For example, if the 16 KB block size is enabled in addition to the default 8 MB block size, and there are six application servers, each with an 8 KB database cache of 2 GB and a 16 KB database cache of 4 GB, you should adjust the data server's 8 KB database cache to ensure that 52 MB (50MB + [12 GB * .01]) is available for control structures, and the 16 KB cache to ensure that 2 MB (24 GB * .005) is available for control structures (rounding up in both cases).

For information about allocating memory to database caches, see Memory and Startup Settings in the "Configuring Caché" chapter of the *Caché System Administration Guide*.

4.3.4 Temporary Globals

Temporary (scratch) globals should be local to the application server, assuming they do not contain data that needs to be globally shared. Often, temporary globals are highly active and write-intensive. This may penalize other ECP application servers sharing the ECP connection if the temporary globals are located on the data server.

4.3.5 Multiple ECP Channels

InterSystems strongly discourages establishing multiple duplicate ECP channels between an application server and a database server to try to increase *bandwidth*. You run the dangerous risk of having locks and updates for a single *logical transaction* arrive out-of-sync on the database server, which may result in data inconsistency.

4.3.6 Load-balanced Application Servers

Connecting users to application servers in a round-robin or load-balancing scheme may diminish the benefit of caching on the application server. This is particularly true if users work in functional groups that have a tendency to read the same data. As these users are spread among application servers, each application server may end up requesting exactly the same data from the data server, which could lead to increased block invalidation as blocks are modified on one application server and refreshed across other application servers. This is somewhat subjective, but someone very familiar with the application characteristics should consider this possible condition.

4.3.7 Repeated References to Undefined Globals

Repeated references to a global that is not defined (for example, **`$Data(^x(1))`** where ^*x* is not defined) always requires a network operation to test if the global is defined on the ECP data server.

By contrast, repeated references to undefined nodes within a defined global (for example, **`$Data(^x(1))`** where any other node in ^*x* is defined) does not require a network operation once the relevant portion of the global (^*x*) is in the application server cache.

This behavior differs significantly from that of a non-networked application. With local data, repeated references to the undefined global are highly optimized to avoid unnecessary work. Designers porting an application to a networked environment may wish to review the use of globals that are sometimes defined and sometimes not. Often it is sufficient to make sure that some other node of the global is always defined.

4.3.8 The $Increment Function and Application Counters

A common operation in online transaction processing systems is generating a series of unique values for use as record numbers or the like. In a typical relational application, this is done by defining a table that contains a "next available" counter value. When the application needs a new identifier, it locks the row containing the counter, increments the counter value, and releases the lock. Even on a single-server system, this becomes a concurrency bottleneck: application processes spend more and more time waiting for the locks on this common counter to be released. In a networked environment, it is even more of a bottleneck at some point.

Caché addresses this by providing the **$Increment** function, which automatically increments a counter value (stored in a global) without any need of application-level locking. Concurrency for **$Increment** is built into the Caché database engine as well as ECP, making it very efficient for use in single-server as well as in distributed applications.

Applications built using the default structures provided by Caché objects (or SQL) automatically use **$Increment** to allocate object identifier values.

$Increment is a synchronous operation involving journal synchronization when executed over ECP. For this reason, **$Increment** over ECP is a relatively slow operation, especially compared to others which may or may not already have data cached (either in the application server buffer pool or the database server buffer pool). The impact of this may be even greater in a mirrored environment due to network latency between the failover members. For this reason, it may be useful to redesign an application to assign a batch of new values to each application server and use **$Increment** with that local batch within each application server, involving the database server only when a new batch of values is needed. (This approach cannot be used, however, when consecutive application counter values are required.) The **$Sequence** function can also be helpful in this context, as an alternative to or used in combination with **$Increment**.

4.4 ECP-related Errors

There are several runtime errors that can occur on a system using ECP. An ECP-related error may occur immediately after a command is executed or, in the case of commands that are asynchronous in nature, such as **Kill**, the error occurs a short time after the command completes.

4.4.1 <NETWORK> Errors

A <NETWORK> error indicates that an error has occurred that could not be handled by the normal ECP recovery mechanism.

In an application, it is always acceptable to halt a process or roll back any pending work whenever a <NETWORK> error is received. Some <NETWORK> errors are essentially fatal error conditions. Others indicate a temporary condition that might clear up soon. However, an expected programming practice is to always roll back any pending work in response to a <NETWORK> error and start the current transaction over again from the beginning.

A <NETWORK> error on a get-type request such as **$Data** or **$Order** can often be retried manually rather than simply rolling back the transaction immediately. ECP tries to avoid giving a <NETWORK> error that would lose data, but gives an error more freely for requests that are read-only.

4.4.2 Rollback Only Condition

The application-side *rollback-only* condition occurs if the data server detects a network failure during a transaction initiated by the application server. It enters a state where all network requests are met with errors until the transaction is rolled back.

A

ECP Recovery Guarantees and Limitations

This appendix describes the guarantees Caché provides during the recovery of an ECP-configured system and the limitations associated with those guarantees. The semantics described are the same as those the global module provides in other contexts, such as in a single-server system or in a Caché cluster. It is divided into the following sections:

- ECP Recovery Guarantees
- ECP Recovery Limitations

A.1 ECP Recovery Guarantees

During the recovery of an ECP-configured system, Caché guarantees the following recoverable semantics:

- In-order Updates Guarantee
- ECP Lock Guarantee
- Clusters Lock Guarantee
- Rollback Guarantee
- Commit Guarantee
- Transactions and Locks Guarantee
- ECP Rollback Only Guarantee
- ECP Transaction Recovery Guarantee
- ECP Lock Recovery Guarantee
- $Increment Ordering Guarantee
- ECP Sync Method Guarantee

In the description of each guarantee the first paragraph describes a specific condition. Subsequent paragraphs describe the data guarantee applicable to that particular situation.

In these descriptions, Process A, Process B and so on refer to processes attempting update globals on an ECP data server. These processes may originate on the same or different application servers, or on the data server itself; in some cases the origins of processes are specified, in others they are not germane.

A.1.1 In-order Updates Guarantee

Process A updates two data elements sequentially, first global ^*x* and next global ^*y*, where ^*x* and ^*y* are located on the same data server.

If *Process B* sees the change to ^*y*, it also sees the change to ^*x*. This guarantee applies whether or not *Process A* and *Process B* are on the same application server as long as the two data items are on the same data server and the data server remains up.

Process B's ability to view the data modified by *Process A* does not ensure that **Set** operations from *Process B* are restored after the **Set** operations from *Process A*. Only a **Lock** or a **$Increment** operation can ensure proper ordering of competing **Set** and **Kill** operations from two different processes during cluster failover or cluster recovery.

See the Loose Ordering in Cluster Failover or Restore limitation regarding the order in which competing **Set** and **Kill** operations from separate processes are applied during cluster dejournaling and cluster failover.

Important: This guarantee does not apply if the data server crashes, even if ^*x* and ^*y* are journaled. See the Dirty Data Reads for ECP Without Locking limitation for a case in which processes that fit this description can see dirty data that never becomes durable before the data server crash.

A.1.2 ECP Lock Guarantee

Process B on DataDataServer S acquires a lock on global ^*x*, which was once locked by *Process A*.

Process B can see *all* updates on DataServer S done by *Process A* (while holding a lock on ^*x*). Also, if *Process C* sees the updates done by *Process B* on DataServer S (while holding a lock on ^*x*), *Process C* is guaranteed to also see the updates done by *Process A* on DataServer S (while holding a lock on ^*x*).

Serializability is guaranteed whether or not *Process A*, *Process B*, and *Process C* are located on the same application server or on DataServer S itself, as long as DataServer S stays up throughout.

Important: The lock and the data it protects must reside on the same data server.

A.1.3 Clusters Lock Guarantee

Process B on a cluster member acquires a lock on global ^*x* in a clustered database; a lock once held by *Process A*.

Process B sees *all* updates to any clustered database done by *Process A* (while holding a lock on ^*x*).

Additionally, if *Process C* on a cluster member sees the updates on a clustered database made by *Process B* (while holding a lock on ^*x*), *Process C* also sees the updates made by *Process A* on any clustered database (while holding a lock on ^*x*).

Serializability is guaranteed whether or not *Process A*, *Process B*, and *Process C* are located on the same cluster member, and whether or not any cluster member crashes.

Important: See the Dirty Data Reads When Cluster Slave Crashes limitation regarding transactions on one cluster member seeing dirty data from a transaction on a cluster member that crashes.

A.1.4 Rollback Guarantee

Process A executes a **TStart** command, followed by a series of updates, and either halts before issuing a **TCommit**, or executes a **TRollback** before executing a **TCommit**.

All the updates made by *Process A* as part of the transaction are rolled back in the reverse order in which they originally occurred.

Important: See the rollback-related limitations: Conflicting, Non-Locked Change Breaks Rollback, Journal Discontinuity Breaks Rollback, and Asynchronous TCommit Converts to Rollback for more information.

A.1.5 Commit Guarantee

Process A makes a series of updates on DataServer S and halts after starting the execution of a **TCommit**.

On each DataServer S that is part of the transaction, the data modifications on DataServer S are either committed or rolled back. If the process that executes the **TCommit** has the **Perform Synchronous Commit** property turned on (`SynchCommit=1`, in the configuration file) and the **TCommit** operation returns without errors, the transaction is guaranteed to have durably committed on all the data servers that are part of the transaction.

Important: If the transaction includes updates to more than one server (including the local server) and the **TCommit** cannot complete successfully, some servers that are part of the transaction may have committed the updates while others may have rolled them back.

A.1.6 Transactions and Locks Guarantee

Process A executes a **TStart** for *Transaction T*, locks global ^*x* on DataServer S, and unlocks ^*x* (unlock does not specify the "immediate unlock" lock type).

Caché guarantees that the lock on ^*x* is not released until the transaction has been either committed or rolled back. No other process can acquire a lock on ^*x* until *Transaction T* either commits or rolls back on DataServer S.

Once *Transaction T* commits on DataServer S, *Process B* that acquires a lock on ^*x* sees changes on DataServer S made by *Process A* during *Transaction T*. Any other process that sees changes on DataServer S made by *Process B* (while holding a lock on ^*x*) sees changes on DataServer S made by *Process A* (while executing *Transaction T*). Conversely, if *Transaction T* rolled back on DataServer S, a *Process B* that acquires a lock on ^*x*, sees *none* of the changes made by *Process A* on DataServer S.

Important: See the Conflicting, Non-Locked Change Breaks Rollback limitation for more information.

A.1.7 ECP Rollback Only Guarantee

Process A on AppServer C makes changes on DataServer S that are part of a *Transaction T*, and DataServer S unilaterally rolls back those changes (which can happen in certain network outages or data server outages).

All subsequent network requests to DataServer S by *Process A* are rejected with `<NETWORK>` errors until *Process A* explicitly executes a **TRollback** command.

Additionally, if any process on AppServer C completes a network request to DataServer S between the rollback on DataServer S and the **TCommit** of *Transaction T* (AppServer C finds out about the rollback-only condition before the **TCommit**), *Transaction T* is guaranteed to roll back on *all* data servers that are part of *Transaction T*.

A.1.8 ECP Transaction Recovery Guarantee

An ECP data server crashes in the middle of an application server transaction, restarts, and completes recovery within the application server recovery timeout interval.

The transaction can be completed normally without violating any of the described guarantees. The data server does not perform any data operations that violate the ordering constraints defined by lock semantics. The only exception is the **$Increment** function (see the ECP and Clusters $Increment Limitation section for more information). Any transactions that cannot be recovered are rolled back in a way that preserves lock semantics.

Important: Caché expects but does not guarantee that in the absence of continuing faults (whether in the network, the data server, or the application server hardware or software), all or most of the transactions pending into an ECP data server at the time of a data server outage are recovered.

A.1.9 ECP Lock Recovery Guarantee

DataServer S has an unplanned shutdown, restarts, and completes recovery within the recovery interval.

The ECP Lock Guarantee still applies as long as all the modified data is journaled. If data is not being journaled, updates made to the data server before the crash can disappear without notice to the application server. Caché no longer guarantees that a process that acquires the lock sees all the updates that were made earlier by other processes while holding the lock.

If DataServer S shuts down gracefully, restarts, and completes recovery within the recovery interval, the ECP Lock Guarantee still applies whether or not data is being journaled.

Updates that are part of a transaction are always journaled; the ECP Transaction Recovery Guarantee applies in a stronger form. Other updates may or may not be journaled, depending on whether or not the destination global in the destination database is marked for journaling.

A.1.10 $Increment Ordering Guarantee

The **$Increment** function induces a loose ordering on a series of **Set** and **Kill** operations from separate processes, even if those operations are not protected by a lock.

For example: *Process A* performs some **Set** and **Kill** operations on DataServer S and performs a **$Increment** operation to a global ^*x* on DataServer S. *Process B* performs a subsequent **$Increment** to the same global ^*x*. Any process, including *Process B*, that sees the result of *Process B* incrementing ^*x*, sees all changes on DataServer S that *Process A* made before incrementing ^*x*.

Important: See the ECP and Clusters $Increment Limitation section for more information.

A.1.11 ECP Sync Method Guarantee

Process A updates a global located on Data Server S, and issues a **$system.ECP.Sync()** to S. Process B then issues a **$system.ECP.Sync()** to S. Process B can see all updates performed by Process A on Data Server S prior to its **$system.ECP.Sync()** call.

$system.ECP.Sync() is relevant only for processes running on an application server. If either process A or B are running on DataServer S itself, then that process does not need to issue a **$system.ECP.Sync()**. If both are running on DataServer S then neither needs **$system.ECP.Sync**, and this is simply the statement that global updates are immediately visible to processes running on the same server.

Important: **$system.ECP.Sync()** does not guarantee durability; see the Dirty Data Reads in ECP without Locking limitation.

A.2 ECP Recovery Limitations

During the recovery of an ECP-configured system, there are the following limitations to the Caché guarantees:

- ECP and Clusters $Increment Limitation
- ECP Cache Liveness Limitation

A.2.1 ECP and Clusters $Increment Limitation

If an ECP data server crashes while the application server has a **$Increment** request outstanding to the data server and the global is journaled, Caché attempts to recover the **$Increment** results from the journal; it does not re-increment the reference.

A.2.2 ECP Cache Liveness Limitation

In the absence of continuing faults, application servers observe data that is no more than a few seconds out of date, but this is not guaranteed. Specifically, if an ECP connection to the data server becomes nonfunctional (network problems, data server shutdown, data server backup operation, and so on), the user process may observe data that is arbitrarily stale, up to an application server connection-timeout value. To ensure that data is not stale, use the **Lock** command around the data-fetch operation, or use **$system.ECP.Sync**. Any network request that makes a round trip to the data server updates the contents of the application server ECP network cache.

A.2.3 ECP Routine Revalidation Limitation

If an application server downloads routines from an ECP data server and the data server restarts (planned or unplanned), the routines downloaded from the data server are marked as if they had been edited.

Additionally, if the connection to the data server suffers a network outage (neither application server nor data server shuts down), the routines downloaded from the data server are marked as if they had been edited. In some cases, this behavior causes spurious `<EDITED>` errors as well as `<ERRTRAP>` errors.

A.2.4 Conflicting, Non-Locked Change Breaks Rollback

In Caché, the **Lock** command is only advisory. If *Process A* starts a transaction that is updating global ^*x* under protection of a lock on global ^*y*, and another process modifies ^*x* without the protection of a lock on ^*y*, the rollback of ^*x* does not work.

On the rollback of **Set** and **Kill** operations, if the current value of the data item is what the operation set it to, the value is reset to what it was before the operation. If the current value is different from what the specific **Set** or **Kill** operation set it to, the current value is left unchanged.

If a data item is sometimes modified inside a transaction, and sometimes modified outside of a transaction and outside the protection of a **Lock** command, rollback is not guaranteed to work. To be effective, locks must be used everywhere a data item is modified.

A.2.5 Journal Discontinuity Breaks Rollback

Rollback depends on the reliability and completeness of the journal. If something interrupts the continuity of the journal data, rollbacks do not succeed past the discontinuity. Caché silently ignores this type of transaction rollback.

A journal discontinuity can be caused by executing **^JRNSTOP** while Caché is running, by deleting the Write Image Journal (WIJ) file after a Caché shutdown and before restart, or by an I/O error during journaling on a system that is not set to freeze the system on journal errors.

A.2.6 ECP Can Miss Error After Recovery

A **Set** or **Kill** operation completes on a data server, but receives an error. The data server crashes after completing that packet, but before delivering that packet to the application server system.

ECP recovery does not replay this packet, but the application server has not found out about the error; resulting in the application server missing **Set** or **Kill** operations on the data server.

A.2.7 Partial Set or Kill Leads to Journal Mismatch

There are certain cases where a **Set** or **Kill** operation can be journaled successfully, but receive an error before actually modifying the database. Given the particular way rollback of a data item is defined, this should not ever break transaction rollback; but the state of a database after a journal restore may not match the state of that database before the restore.

A.2.8 Loose Ordering in Cluster Failover or Restore

Cluster dejournaling is loosely ordered. The journal files from the separate cluster members are only synchronized wherever a lock, a **$Increment**, or a journal marker event occurs. This affects the database state after either a cluster failover or a cluster crash where the entire cluster must be brought down and restored. The database may be restored to a state that is different from the state just before the crash. The $Increment Ordering Guarantee places additional constraints on how different the restored database can be from its original form before the crash.

Process B's ability to view the data modified by *Process A* does not ensure that **Set** operations from *Process B* are restored after the **Set** operations from *Process A*. Only a **Lock** or a **$Increment** operation can ensure proper ordering of competing **Set** and **Kill** operations from two different processes during cluster failover or cluster recovery.

A.2.9 Dirty Data Reads When Cluster Slave Crashes

A cluster slave *Member A* completes updates in *Transaction T1*, and that system commits that transaction, but in non-synchronous transaction commit mode. *Transaction T2* on a different cluster *Member B* acquires the locks once owned by *Transaction T1*. Cluster slave *Member A* crashes before all the information from *Transaction T1* is written to disk.

Transaction T1 is rolled back as part of cluster failover. However, *Transaction T2* on *Member B* could have seen data from *Transaction T1* that later was rolled back as part of cluster failover, despite following the rules of the locking protocol. Additionally, if *Transaction T2* has modified some of the same data items as *Transaction T1*, the rollback of *Transaction T1* may fail because only some of the transaction data has rolled back.

A workaround is to use synchronous commit mode for transactions on the cluster slave *Member A*. When using synchronous commit mode, *Transaction T1* is durable on disk before its locks are released, so *Transaction T1* is not rolled back once the application sees that it is complete.

A.2.10 Dirty Data Reads in ECP Without Locking

If an incoming ECP transaction reads data without locking, it may see data that is not durable on disk which may disappear if the data server crashes. It can only see such data when the data location is set by other ECP connections or by the local data server system itself. It can never see nondurable data that is set by this connection itself. There is no possibility of seeing nondurable data when locking is used both in the process reading the data and the process writing the data. This is a violation of the In-order Updates Guarantee and there is no easy workaround other than to use locking.

A.2.11 Asynchronous TCommit Converts to Rollback

If the data server side of a transaction receives an asynchronous error condition, such as a `<FILEFULL>`, while updating a database, and the application server does not see that error until the **TCommit**, the transaction is automatically rolled back on the data server. However, rollbacks are synchronous while **TCommit** operations are usually asynchronous because the rollback will be changing blocks the application server should be notified of before the application server process surrenders any locks.

The data server and the database are fine, but on the application server if the locks get traded to another process he may see temporarily see data that is about to be rolled back. However, the application server does not usually do anything that causes asynchronous errors