X·IPC Message Oriented Middleware

Version 3

Scalable Message Oriented Middleware
for
Distributed Computing

Product Overview

ENVOY Technologies

X•IPC
Product Overview

X•IPC is an advanced software toolset for the development of multitasking and distributed applications. X•IPC provides fault-tolerant management of guaranteed delivery and real-time message queuing, synchronization semaphores and shared memory, all of which are network-transparent.

With X•IPC, you can:

> ***Speed** your processing–and your business processes–with rapid and guaranteed message delivery and data synchronization services that support critical real-time and mission-critical applications, including telecommunications routing, high-speed message switching, manufacturing process control, transportation systems control, robotics and more.*
>
> ***Simplify** the development, testing, delivery and maintenance of electronic commerce, management and financial applications with a single platform-independent API as well as real-time monitoring and debugging tools that shorten the developer and user learning curves and accelerate delivery of multitasking and distributed applications by as much as 50%.*
>
> ***Interoperate** and insure portability over the widest range of operating system environments, protocols and languages in the distributed processing arena.*
>
> ***Scale** your environment–from stand-alone through networking paradigms–without changing code, as your applications and business activities evolve in size and complexity.*
>
> ***Rely** on your network application communications with guaranteed store and forward messaging, sophisticated exception handling and system fault-tolerance.*

X•IPC is the network-transparent middleware of choice, providing the most comprehensive, scalable, easy to use, rapid and reliable set of services available.

Envoy Technologies Inc.
555 Route 1 South, Fourth Floor, Iselin, NJ 08830
Tel. (732) 636-4700, Fax. (732) 636-4884

Introduction

The vast majority of computer applications under development today comprise multiple processes (or threads) of execution operating in concert with one another to perform the application's objectives. These are referred to as multitasking (or multithreaded) applications when all the processes/threads reside on a single platform, or as distributed applications when some of the processes/threads are separated by a network. The communication among these multiple processes is called Interprocess Communication (IPC).

Interprocess communication (IPC) is at the heart of all multi-tasking client/server, peer-to-peer, multi-tier and other forms of distrib-uted applications.

IPC is at the heart of client/server, peer-to-peer, multi-tier and other forms of distributed applications. In fact, IPC is required for almost any application employing two or more cooperating processes. It is not surprising, then, that the selection of an IPC mechanism can greatly influence the measure of success in the development, deployment and execution performance of such applications.

X•IPC takes the standard IPC techniques, extends their functionality dramatically and makes them network transparent.

The major operating systems provide IPC facilities that support process-to-process communication on a single platform. Although these facilities may provide a relatively complete set of services (message queuing, semaphores and memory sharing), they are confined by the walls of the platform. Far more limited facilities are available for supporting process-to-process communication over a network. Developers must choose between these two facilities for supporting their application's IPC functionality, based on process location.

*X•IPC, by contrast, applies the standard programming model for multitasking IPC (message queues, semaphores, shared memory), expands it to include guaranteed message delivery and makes this enhanced model completely operating system and network transparent.

X•IPC's use of the standard IPC techniques, shortens the learning curve and speeds development.

To the *X•IPC* programmer, the network appears as a single multitasking operating system. With the addition of high-level functionality and extensive monitoring and debugging aids, *X•IPC* facilitates the rapid development and deployment of sophisticated multitasking, network-transparent and enterprise-scalable distributed computer applications.

What is X•IPC ?

X•IPC makes the network appear like a single multitasking environment.

X•IPC is a middleware tool that treats IPC programming as a single problem, regardless of the location of the involved processes and regardless of the variety of operating systems, network protocols and hardware platforms being used. X•IPC essentially permits the developer to conceptualize the programming environment as a virtual multitasking environment where processes communicate and interact with each other using a *single* high-level programming model, independent of the physical location of the processes.

X•IPC insulates an application from its underlying environment.

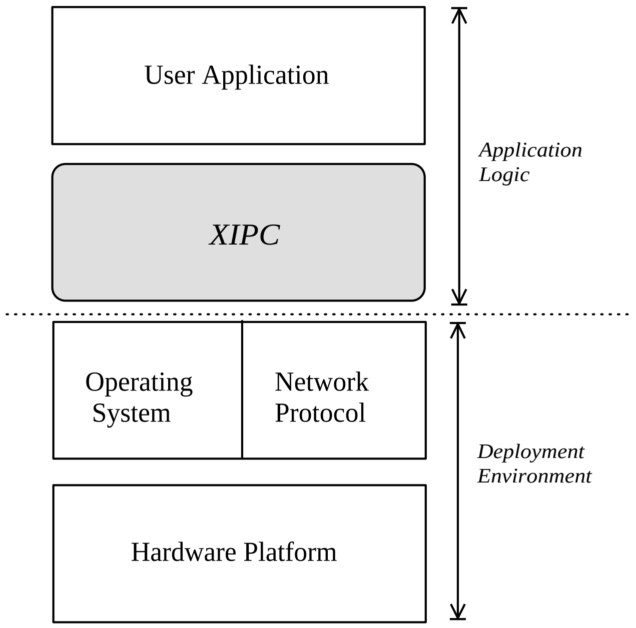
X•IPC's unique IPC approach effectively disengages the programming logic of an application from the issues of application topology and process deployment. The programming logic governing the interaction between processes within an application is unrelated to the positioning of the processes and is unaffected by their occasional repositioning.

"X•IPC makes it possible to create distributed applications without the need for any network level programming." ***DATA COMMUNICATIONS***

X•IPC lies between an application and its underlying environment, providing a rich Application Program Interface (API) that:

- Performs all inter-process communication operations
- Is independent of the underlying operating system, network protocol and hardware platform
- Is easy to learn and use
- Insulates the applications built with X•IPC from changes to the underlying deployment environment.

"Faster time to market...Ease of testing and tuning...Premium performance... X•IPC works...It's reliable." Humphrey Bryson, ***NASDAQ***

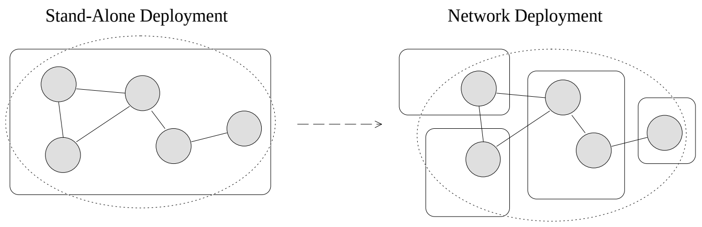


XIPC middleware

The X•IPC programming model has a number of important advantages:

- *Productivity:* With its familiar multitasking model, X•IPC eliminates the need for network programming skills. As a result, the cost of developing and maintaining distributed applications is significantly reduced.
- *Connectivity*: X•IPC's programming model allows processes of a distributed application to appear local to one another – client to server, peer to peer. The processes interact as if there were no network.

"X•IPC cut development time by two-thirds, because it was easy to program."
Pat O'Leary, American Airlines, as reported in ***BYTE*** *Magazine*



X•IPC provides network-transparent connectivity

- *Interoperability:* The X•IPC programming model is preserved even when processes are spread over a network of heterogeneous computer platforms. Application logic is independent of application environment mix.
- *Portability:* With its portable API, provides for immediate source code portability between dissimilar platforms. The IPC portion of an application need only be written once.
- *Scalability:* Applications written using X•IPC are inherently scalable. Applications involving thousands of users and network nodes may be built and quickly deployed. Subsequent adding and deleting of nodes can occur "on the fly" – without disrupting or modifying any aspect of the running application.
- *Reliability*: X•IPC's sophisticated exception handling, fault-tolerant recovery mechanisms and real-time monitoring tools provide the means for building distributed applications that are robust and reliable.

X•IPC allows you to connect multiple clients to multiple servers over complex heterogeneous networks.

Applications built with X•IPC can be scaled to thousands of machines without changing source code.

Who Can Benefit from Using X•IPC ?

"Properly applied, X•IPC has the potential to reduce the time, cost and technical risk of developing complex applications."
The Gartner Group

The Gartner Group has noted, *"Leading edge IS departments and ISVs with the need to develop sophisticated cooperative applications should consider X•IPC for their program to program functions. Properly applied, X•IPC has the potential to reduce the time, cost and technical risk of developing complex applications."*

X•IPC can be employed for constructing elegant solutions to a wide range of distributed computing applications.

Examples of applications that are well-suited for X•IPC include:

- High performance and guaranteed delivery messaging systems
- Asynchronous store-and-forward message-based applications
- Message replication and multi-casting
- Multi-tier client/server applications
- Process control or real-time applications
- Unit-of-work messaging semantics

HIGH PERFORMANCE AND GUARANTEED DELIVERY MESSAGING APPLICATIONS

X•IPC provides guaranteed message delivery between all types of user applications, regardless of operating system or platform volatility, making it the product of choice for handling fail-safe messaging applications.

X•IPC's high performance message queuing mechanisms provide the means for implementing real-time messaging and transaction routing capabilities within stand-alone or distributed applications. X•IPC's network-transparent programming model allows such functionality to operate uniformly, regardless of the location of participating processes – local to one another or distributed over a network. This flexibility is of particular importance when deployment scenarios for the application are either unknown at the time of the application's development or are intended to vary.

For applications in need of particularly high-throughput messaging, X•IPC provides a "burst-mode" capability that transfers messages at extremely high rates by employing protocol optimizing read-ahead mechanisms.

X•IPC additionally provides guaranteed message delivery between all types of user applications, regardless of operating system or platform volatility, making it the product of choice for handling fail-safe messaging applications. These messaging capabilities are complemented by features that support advanced flow-control functionality, including:

- *Tailored message tracking:* X•IPC lets you determine how far along the data path a message is tracked between originating and destination applications.
- *Process Synchronization:* In conjunction with the message tracking, X•IPC provides the means for programs to synchronize their processing with the forward progress of in-flight messages.
- *Flexible message queue configuration:* X•IPC's message queues are individually sized, can support extremely large capacity requirements and can be read by multiple server programs for load balancing.
- *Traffic management*: X•IPC messaging applications can be structured to react automatically to surges of message traffic by widening the message bandwidth (more queues) or by accelerating message throughput (more processing programs).

X•IPC "burst" mode messaging is ideal for implementing high-volume, high-throughput messaging applications.

ASYNCHRONOUS, STORE AND FORWARD MESSAGING APPLICATIONS

X•IPC's asynchronous messaging functionality provides the means for building distributed applications that support "occasionally connected" user nodes. This is often a requirement within applications having travelling salesmen, or similar laptop-toting users. Insurance applications are a common example of applications with this need.

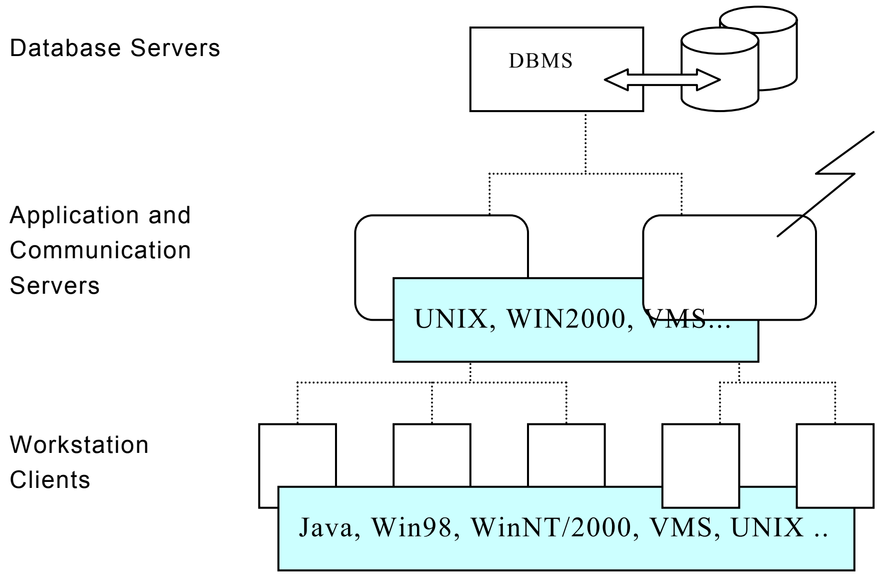
X•IPC's asynchronous messaging functionality provides the means for building distributed applications that support "occasionally connected" user nodes.

A related category of applications – *workflow applications* – can also benefit from the "fire and forget" asynchronous, guaranteed messaging within X•IPC.

MULTI-TIER CLIENT/SERVER APPLICATIONS

The X•IPC programming model is well suited for implementing applications that are logically and/or physically layered as multiple tiers. Each tier supports an autonomous level of functionality within the application and the tiers interact dynamically with each other. These applications generally exhibit the following architectural appearance:

X•IPC is well suited for implementing client/server applications involving heterogeneous systems and/or multiple tiers.


Database Servers
DBMS
Application and Communication Servers
UNIX, WIN2000, VMS...
Workstation Clients
Java, Win98, WinNT/2000, VMS, UNIX ..

Dynamic multi-tier client-server interaction

Such architecture is often associated with legacy applications that are being downsized into client/server distributed environments. Component processes in such architecture must be able to communicate, synchronize and share data with each other in a generally asynchronous manner. X•IPC's rich set of asynchronous capabilities simplifies the logical expression and coding implementation of such requirements.

PROCESS CONTROL AND REAL-TIME APPLICATIONS

X•IPC's wide range of IPC mechanisms and its high performance make it ideal for building real-time applications.

X•IPC's wide range of IPC mechanisms (reliable and high-speed message queuing, semaphores and shared memory) and operation modes (synchronous and asynchronous) position it as an ideal tool for building complex process-control and real-time applications. This is especially true where the interaction between processes is not easily expressed using a simple message-passing paradigm.

Component processes within such applications generally require a more finely granular set of IPC tools for supporting the process-to-process (or thread-to-thread) synchronization requirements of the application. X•IPC's extensive IPC capabilities are ideal for building such systems.

Such implementations have the added feature of being portable among different deployment environments.

X•IPC's easy integration with native operating system event mechanisms additionally allows for the integration of *X•IPC* operations together with system-level events. It is thus possible to have a process wait simultaneously for operating system *and* *X•IPC* activity, using the standard set of operating system event primitives.

An example of such an application would be one in which processes must interoperate with voice recognition equipment while simultaneously communicating with other application processes. It is possible to uniformly multiplex the voice and *X•IPC* data streams using the native operating system facilities.

X•IPC Capabilities

SYNCHRONOUS FUNCTIONALITY

X•IPC's asynchronous functionality allows you to leverage the inherent parallelism of the network environment.

In the flurry of IPC activity occurring within an application, certain operations will potentially block due to the state of the IPC objects involved. For example, a process may issue a request to receive a particular message that has not yet been sent; or a process attempts to write to an area of shared memory that is currently locked by another process; or a process may block while waiting for a group of events to occur.

X•IPC supports three forms of behavior for reacting to these situations synchronously:

- *No Wait:* The operation returns an error code indicating that the desired operation could not be completed at the current time. No blocking occurs.
- *Wait:* The operation blocks the calling process until the operation is able to complete.
- *Time Out:* The operation blocks the calling process for a user-specified period of time, while waiting for the operation to complete.

ASYNCHRONOUS FUNCTIONALITY

X•IPC eliminates the need for threads in achieving asynchronous operation.

An alternative approach for such situations is to have the operation complete asynchronously, in the background, without blocking the mainline logic of the calling process. The benefits of such an approach are many, the most significant being that such a mechanism allows an application's distributed processes to execute *concurrently*.

With X•IPC, it is natural to initiate concurrent operations involving multiple network platforms and to have them run to completion in parallel. This has the direct and positive effect of leveraging the inherent parallelism of a network environment.

Causing an X•IPC operation to run asynchronously is straightforward. The operation is initiated with an X•IPC asynchronous option. The three supported asynchronous options are:

X•IPC is additionally thread-safe and can be used for developing highly scalable asynchronous applications using multiple threads.

- *Callback:* X•IPC notifies the process of the operation's completion by executing a user-specified function.
- *Post:* X•IPC notifies the process of the operation's completion by setting a user-specified X•IPC event semaphore.
- *Ignore:* X•IPC completes the asynchronous operation silently.

X•IPC is additionally thread-safe and can be used for developing highly scalable asynchronous applications using multiple threads.

EVENT-DRIVEN GUI PROGRAMMING

X•IPC 's asynchronous capabilities are powerful tools for incorporating event-driven functionality within distributed applications. This is particularly important when part of the application must interface with an event-driven Graphical User Interface (GUI) such as MS-Windows or X-Windows. X•IPC logic for managing an application's process-to-process communication can be naturally integrated within the event-driven programming environments of such GUIs.

GUARANTEED DELIVERY MESSAGE QUEUING

X•IPC 's guaranteed message delivery eliminates the most complex network programming tasks - the need to build recovery logic into each individual application to recover from network and system failures. This service is handled by X•IPC <u>automatically</u>.

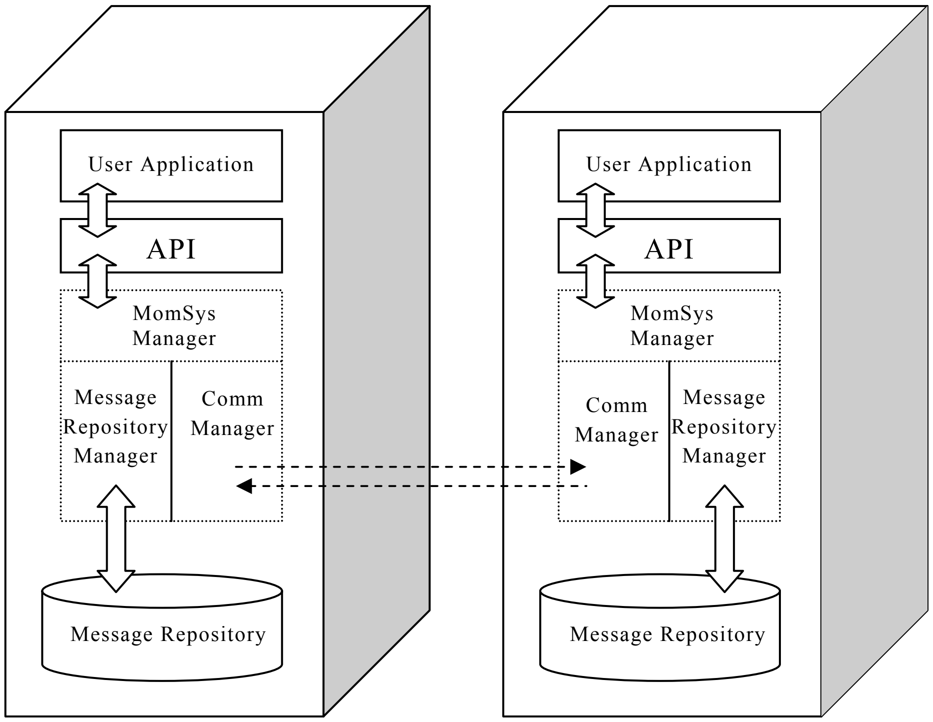
X•IPC 's guaranteed delivery message queuing provides guaranteed store-and-forward message delivery between all types of user applications, regardless of operating system or platform volatility. “Islands” of client/server computing can be easily linked across the enterprise and include remote and mobile users.

In providing guaranteed message delivery, X•IPC provides a sophisticated and powerful replacement for batch file transfers between applications or systems. X•IPC eliminates one of the most pervasive problems with batch file transfer: lack of timeliness resulting from network or system outages. With X•IPC, business application messages are communicated among user applications immediately, as the information is created, eliminating otherwise inherent delays in processing. Examples of such applications include orders, shipping requests, loan applications, insurance claims, etc.

In addition, X•IPC’s guaranteed message delivery eliminates the most complex network programming tasks – the need to build recovery logic into each individual application to recover from network and system failures. This network-transparent service is built into X•IPC, resulting in much shorter development time and much more reliable and robust applications.

The Reliable Messaging Programming Model

The X•IPC reliable messaging model is manifest within the MomSys subsystem of X•IPC . It comprises an API, a Subsystem Manager, a Message Repository Manager and a scalable Communication Manager working as a unit, and communicating with the MomSys subsystems of the other remote instance. The basic MomSys architecture is depicted below.



The MomSys Programming Model

Copies of sent messages are stored locally until they are known to have been successfully delivered.

Message Tracking and Synchronization

Sophisticated message tracking supports automatic message acknowledgments as well as message response synchronization.

The connection between communicating applications can be considered a "data path" that messages follow to their destination. X•IPC enables an application to track each sent message to a specified point along the data path. This is defined as the Message Tracking Level. Three tracking levels can be designated:

- Messages may be tracked until they are stored within the local node's message repository.
- Messages may be tracked until they are stored within the remote node's message repository.

- Messages may be tracked until they are successfully received by a program on the remote node.

X•IPC maintains the current status of in-flight messages as they travel along their data path; this information is available to the message-originating application––synchronously (i.e., by waiting for the message to reach a certain level) or asynchronously (i.e., by posting a callback function or program that should execute when the message reaches a certain level).

An in-flight message that is not successfully delivered to its target by a user-specified time can be automatically expired, and journalled by X•IPC.

Message Expiration and Journalling

X•IPC allows a message expiration time be specified for each sent message. In this way, in-flight application messages that are not successfully delivered by a certain time (typically due to network or system outages) can be identified by X•IPC as having expired. Expired messages can then be automatically removed from the message flow.

Journalling of expired messages is managed automatically by X•IPC for auditing purposes.

"Unit of Work" Messaging Semantics

X•IPC MomSys supports message-based unit-of-work semantics. It is possible to bracket a sequence of messaging operations (i.e., send, receive) as occurring within a single unit-of-work. The bracketed operations may subsequently be committed, as an atomic operation, when the application deems it appropriate or they may be rolled back as having never occurred.

It is possible to bracket a sequence of X•IPC messaging operations (i.e., send, receive)as occurring within a single, atomic unit-of-work.

Message Priorities

X•IPC supports a full range of message priorities to be specified by the sender program for controlling the movement of a message (relative to other in-flight messages) as it travels to its destination queue, and for controlling its positioning relative to other messages within the destination queue. These priorities need not be the same.

Fault-Tolerant, Enterprise Scalable Queue Catalog

X•IPC provides a high-level queue naming and discovery mechanism that is fault-tolerant and is highly scalable. A queue may be positioned anywhere on a network, and then subsequently discovered symbolically by other programs wishing to send messages to the queue.

X•IPC queues, once defined, can be repositioned "on the fly" to another node, without having to stop the application, causing in-flight messages to be dynamically routed to the new location.

Furthermore, queues, once defined, can be repositioned "on the fly" to another node - without having to stop the application – causing in-flight messages to be dynamically routed to the new queue location.

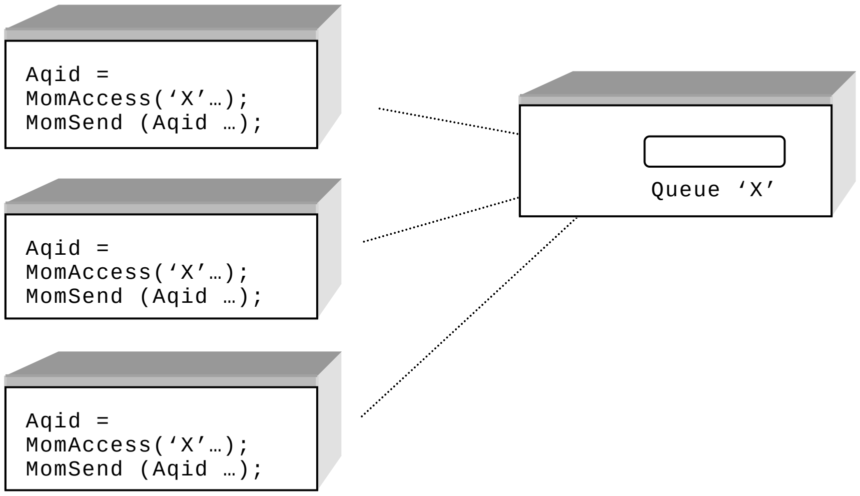
Publish and subscribe groups and filters may also be registered and discovered via the X•IPC catalog.

The X•IPC Catalog is implemented using a fault-tolerant architecture, and may be replicated on multiple servers across the enterprise for enhancing the "locality" of catalog inquiries.

Dynamic Queue Relocation

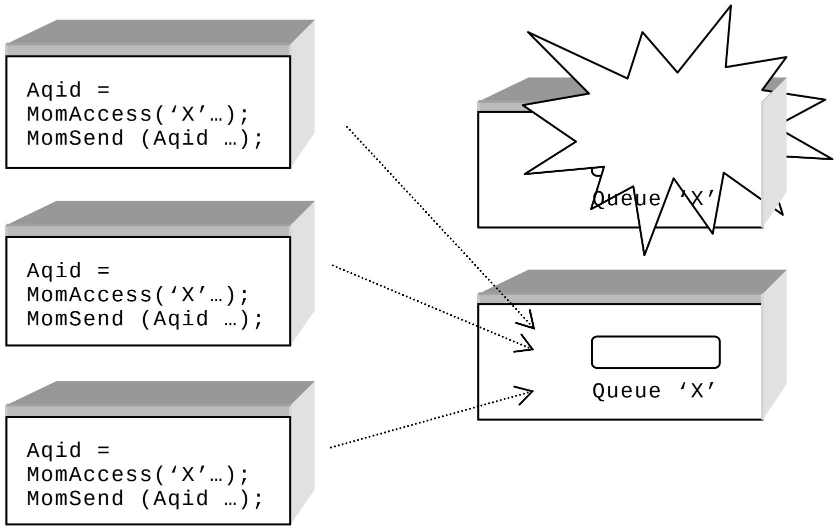
Using X•IPC MomSys, it is possible to relocate a queue from its current location to a new location and to have all programs that are currently sending messages to that queue have their messages *automatically* start going to the queue's new location. This occurs "on the fly," without the sending programs needing to make re-routing coding provisions and without them being aware of the targeted queue's new location.

Consider the following example in which three programs are sending messages to an app-queue that they have accessed, having the well-known queue name 'X.'


```
Aqid =
MomAccess('X'…);
MomSend (Aqid …);
```
```
Aqid =
MomAccess('X'…);
MomSend (Aqid …);
```
```
Aqid =
MomAccess('X'…);
MomSend (Aqid …);
```

MomSys sending messages to Queue 'X'

Now consider what will happen if the server upon which ‘X’ is located crashes. X•IPC allows the user to create a new queue ‘X’ on a new server. This causes all subsequent messages sent to ‘X’ to route to the new ‘X’ automatically. The client programs themselves remain unaware of this “relocating of app-queue ‘X’. Messages are now sent to the new ‘X’:

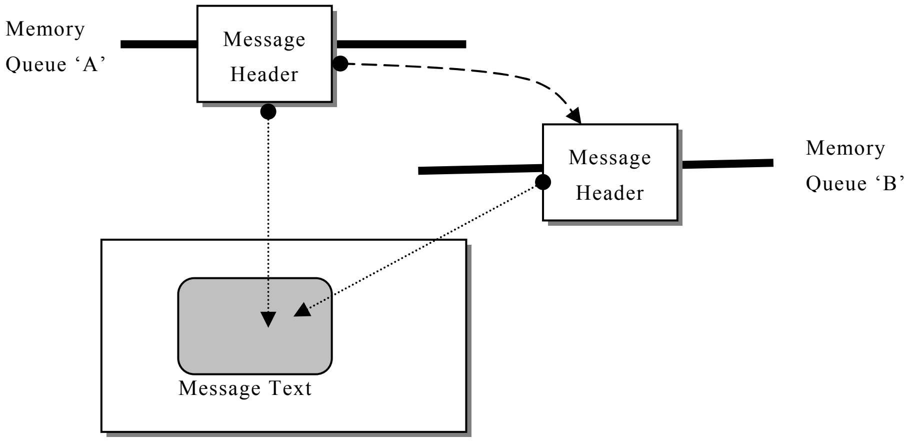


MomSys sends messages to the new 'X' on a new machine automatically

HIGH-PERFORMANCE MEMORY-BASED QUEUING

X•IPC 's message queuing functionality includes support for high-performance memory-based message queuing.

When employed within a single node for supporting intra-nodal IPC, X•IPC memory queuing can move thousands of messages per second between executing processes / threads. In such settings it is also possible to route messages between queues by simply moving the message *headers* from queue to queue without having to read and write message contents.



X•IPC memory queue message routed from queue ‘A’ to queue ‘B’ by moving its header, without touching the message text

Using the above functionality, large messages (such as images) can be efficiently moved between queues, independent of the message size.

Queue "Burst" Throughput

X•IPC memory queues support a high-performance "burst" mode with which it is possible to send messages between machines at extremely high rates One example where this feature is useful is for replicating large databases over a network in near real-time.

X•IPC burst mode queuing achieves its high-throughput performance by pushing message data through the protocol with minimal bandwidth overhead, while also employing advanced read-ahead logic.

Memory Queue Sizing

X•IPC message queues can be created dynamically with individual capacity limits. The limits of memory queues may be specified in terms of maximum message capacity, maximum total byte capacity, both limits or neither. Using this capability it is possible to throttle the flow of message traffic within an application's queues on a queue-by-queue basis, as well as controlling the amount of memory consumed by each queue.

Individual queue sizing provides the architectural flexibility needed for tailoring each message queue for the traffic flow it is intended to support. Real-time communication applications, for example, often require large-capacity message queues that surpass the built-in limits of certain native operating system queuing services. X•IPC queues are not subject to such limits.

Memory Queue Overflow Spooling

Automatic queue overflow spooling is useful for handling real-time message feeds that cannot be throttled without losing messages.

Complementing the specific sizing capability of memory queues is the ability to employ memory queue overflow spooling on an as needed, queue-by-queue basis. When spooling for a queue is active, the memory queue is given elasticity for accepting messages beyond its normal capacity by spooling overflow messages to disk until space on the queue becomes available. At that time, the queue automatically absorbs the spooled messages.

Overflow spooling can be triggered to react automatically to heavy traffic surges. This can be particularly useful for handling real-time message feeds that cannot be throttled without losing messages.

Memory Queue Message Selection

X•IPC provides the developer with considerable flexibility and control in sending and receiving messages to and from memory-based queues. Of particular note is the ability to operate atomically on multiple message queues. For example, you can retrieve the highest priority message, or the oldest message, from across multiple queues. Similarly, X•IPC permits the dispatching of messages onto the shortest of a group of queues, yielding automatic queue balancing.

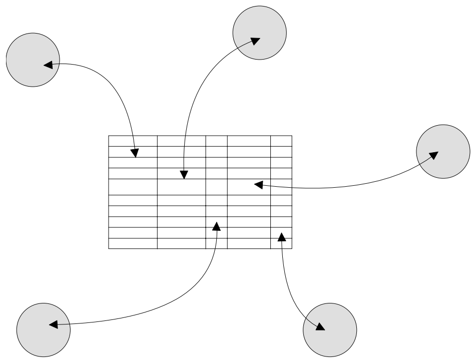
Memory Queue Browsing

X•IPC message queue functionality also includes the ability to momentarily freeze memory queue traffic in order to traverse, examine, prune or even rearrange the messages currently held on a queue. Access to queue data while the data is still queued is a prerequisite for building high-performance message filtering programs. Similar functionality is not available using most native queuing services.

MEMORY SHARING

X•IPC shared memory segments support memory sharing between processes of an application. Using X•IPC shared memory segments, processes are able to share memory-resident data with each other.

X•IPC memory sharing provides an extremely fast, multi-access, memory resident database that is also network transparent.



Network transparent shared memory

Besides supporting the ability to read and write segment data, X•IPC also introduces a number of innovations in managing access to shared memory.

Synchronized Memory Read and Write Operations

X•IPC shared memory read and write operations are guaranteed to execute atomically, regardless of the amount of data involved. X•IPC, as such, enforces serialization of data access to its memory segments. It is thus not necessary to manually enforce serialized access to shared memory for preventing overlapping read and write operations – as is the case with native operating system shared memory services. This is done automatically by *X•IPC*.

Byte-Level Access Controls

X•IPC introduces the notion of shared memory access controls. These mechanisms permit the dynamic imposition of byte-level read or write access limitations over all or parts of *X•IPC* memory segments. Particular areas of segments can be made read-only or entirely inaccessible for long or brief periods of time. This can be used for implementing in-memory data tables requiring such forms of data protection.

Shared Memory Locking

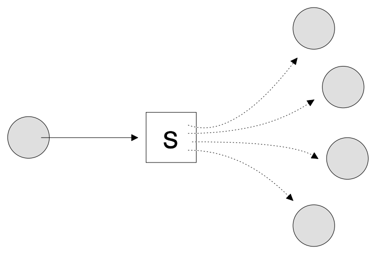
A specific form of access control is that of memory locking. Memory locking can be specified with byte-level granularity to lock all or arbitrary parts of memory segments, on the fly, for performing updates to those areas. Read or write operations attempted against a locked area of memory are prevented until the memory is subsequently unlocked.

Distributed applications that require memory-resident data to be globally available to multiple distributed process, (and for which using a DBMS would be an expensive overkill), can employ *X•IPC*'s memory sharing as an elegant high-performance solution. With *X•IPC*, it is possible to attain many of the data sharing benefits of a DBMS without the accompanying overhead.

SEMAPHORES

Semaphores are ideal for coding synchronization relationships concisely. The setting of a single semaphore can immediately influence the execution of many distributed processes.

X•IPC semaphores are used for supporting event synchronization and resource management occurring among processes of a distributed application. Semaphores are especially useful for expressing interprocess synchronization relationships in a concise manner. The setting of a single semaphore by one process can immediately influence the execution of many other distributed processes.



Network transparent semaphore synchronization

X•IPC supports two classes of semaphores: event semaphores and resource semaphores.

Event Semaphores

X•IPC event semaphores are general-purpose flags that can be *set* or *cleared* in response to the occurrence of application events within a distributed application. Processes waiting for one or more such events to occur wait for the respective semaphores to become set. A process detecting the occurrence of an event notifies the other processes by setting the appropriate semaphore.

Event semaphores can be used for affecting simple and concise forms of one-to-many interprocess communications. By setting a single semaphore, a process can notify hundreds (or thousands) of other processes of an important piece of application-related information for them to react to individually. An example of this is a stock price monitoring system that uses semaphores for notifying all interested parties when selected stocks rise or drop across certain threshold values. This form of leveraged communication between processes is unique to event semaphores.

Resource Semaphores

Resource semaphores are counting mechanisms that control access to limited-supply resources within an application. Processes competing for access to such resources negotiate access to them by means of resource semaphores. Distributed applications that manage facilities or devices over which there is more demand than supply can be designed in a network transparent manner using resource semaphores.

As with other X•IPC facilities, it is possible to arrange that the resource synchronization requirements of an application be performed in an asyn-chronous and network transparent manner. This is often a characteristic of distributed process control applications, where access to physical devices and other equipment must be controlled with network transparency.

Operations Involving Multiple Semaphores

X•IPC provides the means for having a process wait synchronously or asynchronously on a list of events or resources. Such operations can be specified in three forms. When expressed within the context of event semaphores, they are:

- *ANY* - Wait for any of the listed events to occur.
- *ALL* - Wait for all of the listed events to occur cumulatively, over time.
- *ATOMIC* - Wait for all the events to occur simultaneously.

Similar specifications can be made when waiting for lists of X•IPC resource semaphores to become available. X•IPC's ability to group semaphores atomically within single operations provides the means for expressing elegant solutions to complex synchronization requirements.

Semaphores can be used for broadcasting the occurrence of application events.

There are significant advantages in using semaphores to support an application's event synchronization activity, as opposed to employing a messaging mechanism. Building such functionality using messaging tools requires that the application implement its own internal mapping system between messages and events, a system that additionally tracks "who is waiting for what" and, furthermore, remembers the appropriate notification logic for each waiting process. Besides being a non-trivial programming exercise, such an approach is indirect in concept and thus awkward to apply.

X•IPC semaphores, by contrast, provide a concise and direct programming model for implementing an application's interprocess synchronization logic.

TRIGGERS

Triggers are an invaluable tool for building event-driven exception handling into a distributed application.

X•IPC introduces an additional form of asynchronous functionality, that of X•IPC triggers. X•IPC triggers are mechanisms that continuously monitor the state of activity within an application's IPC environment. X•IPC triggers operate asynchronously to the application's logic, only coming into play when the monitored situation occurs.

X•IPC triggers open up a wide range of potent IPC programming techniques. It is possible, for example, to have an application set triggers that notify it when:

- A specific message queue rises above 75% (or some other threshold) of its message capacity. The application can react in a variety of ways, such as enabling queue overflow spooling (in anticipation of an overflow), or by starting more consumer processes to relieve the pressure on the queue.
- A specific area of shared memory is written to. Processes in the application, waiting for such an update, can react by making note of the new data and taking a subsequent designated action.
- A specific process removes a specific message from a queue. The process that produced the message can be automatically notified of the message's arrival at the consuming process.

X•IPC triggers, when combined with the general asynchronous command options, provide the means for building application logic that is entirely event-driven. As noted above, this is especially significant when the application must interface with an event-driven Graphical User Interface.

REAL-TIME MONITORING AND DEBUGGING

X•IPC monitors can cut the cost of testing and debugging a distributed application by 50%.

X•IPC includes full-screen interactive monitors that provide continuous real-time views of the activities occurring within an application's IPC environment. The monitoring facility does not require any special preparation of the application being monitored. It can be invoked to examine the IPC status of production applications in the field, without any extra provisions and without incurring performance overhead in the application when monitoring is not in use.

When invoked, monitoring becomes an invaluable tool for identifying IPC problems occurring within an application. Users of *X•IPC* have described its real-time monitors as being worth their entire investment in *X•IPC* technology.

X•IPC's monitors provide critical support to distributed application developers throughout the application life cycle.

Application Development

The benefits of developing distributed applications with *X•IPC* and its monitoring tools are best measured in terms of development time saved. The ability to watch a network application execute with all of its IPC activities exposed – and to have it do so in slow motion – is of immeasurable value when tracking down elusive coding bugs. The total development time for distributed applications can be cut significantly by means of *X•IPC*'s monitors alone.

Application Testing

"X•IPC's real-time monitors and debuggers are worth our entire investment in X•IPC ."
Bill Cason, VP Engineering at ***IBM/Soft*Switch***

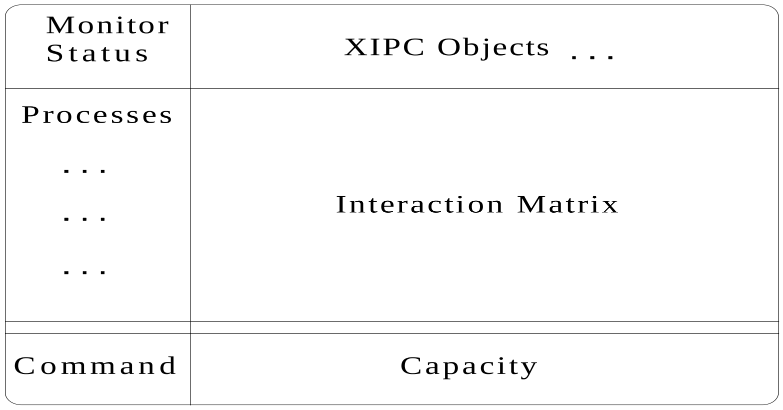
The extent to which an application's performance is tested and analyzed during its test period is usually an accurate predictor of how reliably the application will run in the field. *X•IPC*'s monitoring tools provide detailed information regarding an application's IPC activity for studying the strong and weak links within an application's architecture. High-water marks, low-water marks, asynchronous operation backlogs, text fragmentation and message throughput characteristics are examples of information available using the monitors.

Application Maintenance

X•IPC monitors are support tools as well. Applications built using *X•IPC* are inherently monitorable *after* they are shipped. No longer is it necessary to recreate problems on special "debug" versions of an application in order to study the problems. *X•IPC*-based applications can be studied wherever they are, as soon as they begin to exhibit unexpected behavior.

X•IPC-based production applications can be studied remotely, as soon as they begin to exhibit unexpected behavior.

X•IPC monitors have a spreadsheet "look and feel." Information appears in a matrix-like display where processes and IPC objects form the axes of the matrix. Interactions between processes and IPC objects are displayed within the body of the matrix. The following diagram depicts the general layout of an X•IPC monitor screen.



General layout of XIPC real-time monitors

Monitor Modes

X•IPC monitors update the display of ongoing X•IPC activity in one of the following modes:

- *Interval Snapshot Mode:* Refreshes the monitor display at a user-specified millisecond interval.
- *Trace Flow Mode:* Refreshes the monitor after every X•IPC operation and pauses a user-specified number of milliseconds between each X•IPC operation. This permits an application's IPC activity to be viewed in slow motion.
- *Trace Step Mode:* Similar to Trace Flow Mode, except that the monitor completely stops the application after each X•IPC operation. The user presses a key to single-step to the next operation. This mode is ideal for intensive IPC debugging exercises.

Zoom Windows

X•IPC monitors provide zoom windows that permit developers to focus on the activity of specific aspects of an application's IPC environment, while continuing to monitor the general ebb and flow of IPC activity occurring within the application. A wide range of zoom windows is available, including:

- Zoom in on a particular message queue. It is possible to watch the actual message traffic as it moves through a queue.

- Zoom in on a semaphore. It is possible to track all the processes that are waiting for a particular event and to observe the order in which they blocked and will be woken up.
- Zoom in on a particular process. It is possible to watch every IPC operation being performed by a particular process within an application.

Browse Windows

A second, and even more powerful, monitoring mechanism is the monitor's full screen browse windows. Browsing provides a complete and frozen view of all the message queues and memory segments within an application. It is possible to "walk" through all the messages on a queue or to examine the contents of shared memory segments. It is also possible to search for character strings or hexadecimal patterns of data within message queues or shared memory.

It is possible to browse and examine all the messages on a queue, without touching the application.

Using the browse window, it is possible to confirm the format, length, contents, priority, offset and sequencing of messages on a queue, as well as analyzing the contents of shared memory segments.

Watch Windows

The most advanced form of X•IPC monitoring is achieved using the monitor's watch windows. With them it is possible to observe the contents of memory segments changing in real-time, or at a user-specified update rate. It is additionally possible to observe the locking and unlocking of memory segment areas, as they occur in real-time.

With X•IPC monitors, it is possible to observe the contents of shared memory changing in real-time.

INTERACTIVE COMMAND INTERPRETER

X•IPC provides an interactive command language for executing all of X•IPC's operations interactively, or from within an operating system shell. This eliminates the need to write C-language programs in situations when ad-hoc IPC operations are necessary. This capability is particularly useful for testing an application's handling of various IPC scenarios that may be otherwise difficult to create through the normal execution of the application.

Using the interactive interpreter, it is additionally possible to initiate asynchronous X•IPC operations that, upon their completion, kick off operating system commands. One such example would be to have an operating system disk backup utility triggered by the asynchronous arrival of an X•IPC message or by the setting of an X•IPC semaphore.

X•IPC Case Studies

Since its introduction in 1990, usage of X•IPC as the premier tool for distributed application development has grown rapidly. X•IPC is currently deployed in many mission-critical applications worldwide. Selected case studies follow:

NASDAQ

"Faster time to market...Ease of testing and tuning...Premium performance... X•IPC works...It's reliable." Humphrey Bryson, ***NASDAQ***

NASDAQ employed X•IPC for developing the workstations in their "Stock Market of the Next 100 Years." NASDAQ's Humphrey Bryson found that "the advantages of using Momentum's X•IPC product [include]...Faster time to market...Ease of testing and tuning applications...premium performance. X•IPC works. It's an innovative solution to our requirements. It's reliable."

AMERICAN AIRLINES DECISION TECHNOLOGIES (AADT)

"X•IPC has functionality that is an order or two magnitudes more advanced than the native IPC tools." Pat O'Leary, ***AADT***

As part of American Airlines' move to open systems, AADT developed a database server mechanism that removes all database dependencies from an application. Built using X•IPC, the server is now at the hub of American Airlines' cargo routing system. Patrick O'Leary of AADT determined that X•IPC has "functionality that is an order or two magnitudes more advanced than the native IPC tools."

CANADIAN PACIFIC RAILROAD (CP-RAIL)

Noting X•IPC's "impressive" real-time capabilities, ***Application Development Trends*** *awarded* ***CP-Rail*** *and* ***Momentum Software Corp****. its 1997 Innovator Award.*

CP-Rail has relied on X•IPC to build its latest generation of Central Train Control Systems. In addition to guaranteeing message delivery in this mission-critical environment, X•IPC has also reduced response time from 4 seconds to 80 milliseconds. Noting X•IPC's "impressive" real-time capabilities, *Application Development Trends* awarded CP-Rail and Momentum Software Corp. its 1997 Innovator Award.

X•IPC's network transparency provided the programming model needed by CP-Rail system architects to develop an application that not only shares IPC objects independent of process location, but that is efficient as well. Bruce Calder of CP-Rail characterized X•IPC as "a critical factor in improving CP-Rail's train dispatching efficiency. We have to date harnessed about 5% of X•IPC's power."

IBM / SOFT*SWITCH

Soft*Switch, a leading vendor of mainframe-based electronic mail switches, decided to implement its mail switch technology for open system environments using X•IPC. The new switch had to be easily portable and extendible across a range of operating system environments. IBM/Soft*Switch additionally needed advanced IPC functionality for implementing its switch. Bill Cason, VP of Engineering, commented, "X•IPC's real-time monitors and debuggers are worth our entire investment in X•IPC."

In Closing....

X•IPC is supported on more than 25 operating system platforms as well as on the major network protocols. X•IPC has been well received by its users and the trade press.

"Programmers interested in distributed application programming should take a serious look at X•IPC."
IEEE Software Journal

Data Communications praised X•IPC, stating, "In essence, X•IPC makes it possible to create distributed applications without the need for any network level programming." They further commended X•IPC's singular approach towards intra-nodal and inter-nodal IPC: *"This hybrid approach greatly reduces programming effort and allows programmers to gracefully move their current algorithms to distributed, multiplatform environments."*

The *IEEE Software Journal* concluded its review of X•IPC with the recommendation that, *"Programmers interested in distributed application programming should take a serious look at X•IPC."*