Envoy Connect XIPC Connector

Version 3.4.0

Envoy Connect XIPC Reference Manual

Envoy Technologies Inc.

555 Route 1 South
Iselin, NJ 08830

http://www.envoytech.com

X⅄IPC VERSION 3.4.0

REFERENCE MANUAL

TABLE OF CONTENTS

1. INTRODUCTION

This document is a Reference Manual for *X•IPC* 's *system-level* commands, functions and macros, all of which are listed below. This volume also contains other important reference information, specifically error codes (by symbolic error name and by message number) and user data structures. Sample programs for all *X•IPC* subsystems will be found in the respective subsystem Reference Manuals.

1.1 *X•IPC* Environmental Variables, Commands, Functions and Macros

X•IPC Environment Variable List

`XIPCROOT`	*Platform Directory Environment Variable (network, local and stand-alone*)
`XIPC`	*Instance Name Environment Variable (network, local and stand-alone*)
`XIPCCAT`	*Catalog Node Environment Variable (network only)*
`XIPCCATLIST`	*Catalog Node List Environment Variable (network only)*
`XIPCHOST`	*Instance Node Environment Variable (network only)*
`XIPCHOSTLIST`	*Instance Node List Environment Variable (network only)*

X•IPC Command List

Platform Commands

`xipcinit`	*Initiate X•IPC Environment on Platform*
`xipcterm`	*Terminate X•IPC Environment on Platform*

Instance Commands

`xipcstart`	*Start an X•IPC Instance*
`xipclist`	*List Active Network Instances*
`xipcstop`	*Stop an X•IPC Instance*

`xipc` - *The X•IPC Interactive Command Interpreter*

Note that the subsystem specific interactive commands (for MomSys, QueSys, MemSys and SemSys) are defined in the respective Reference Manuals.

General Interactive Commands

`!`	*Execute Operating System Command*
`acb`	*Display Contents of ACB*
`callback`	*Assign Callback Command*
`help`	*Display List of Arguments*
`quit`	*Logout and Quit*
`uid`	*Display Current User ID*

Other Interactive Commands

`xipcabort`	*Abort a User*
`xipcconnect`	*Connect to a Login*
`xipcdisconnect`	*Disconnect from a Login*
`xipcerror`	*Display Error Message*
`xipcfreeze`	*Freeze Instance*
`xipcidlewatch`	*Control Idle Watch Monitoring*
`xipcinfologin`	*Get Login Information*
`xipcinfoversion`	*Get Xipc Version Information*
`xipcinit`	*Initiate Xipc Platform Environment*
`xipclist`	*List Active Network Instances*
`xipclogin`	*Log Into an Instance*
`xipclogout`	*Log Out of Instance*
`xipcmasktraps`	*Activate Trap Masks*
`xipcstart`	*Start an Instance*
`xipcstop`	*Stop an Instance*

`xipcterm`	*Terminate Xipc Platform Environment*
`xipcunfreeze`	*Unfreeze Instance*
`xipcunmasktraps`	*Deactivate Trap Mask*

X•IPC Function List

Note that the subsystem-specific functions (for MomSys, QueSys, MemSys and SemSys) are defined in the respective Reference Manuals.

`XipcAbort()`	*Abort a User by Forcing a Log Out*
`XipcConnect()`	*Connect to a Login*
`XipcDisconnect()`	*Disconnect from the Current Login*
`XipcError()`	*XIPC Error Code Translation Function*
`XipcFreeze()`	*Freeze an Instance*
`XipcGetOpt()`	*Get Network Timeout Detection Parameters*
`XipcIdleWatch()`	*Control Idle Watch Monitoring*
`XipcInfoLogin()`	*Get Login Information*
`XipcInfoSystemError()`	*Get Additional System Error Information*
`XipcInfoUser()`	*Get User Information*
`XipcInfoVersion()`	*Get XIPC Version Information*
`XipcInit()`	*Initiate XIPC Platform Environment*
`XipcList()`	*List Active Network Instances*
`XipcLogin()`	*Log Into an Instance*
`XipcLogout()`	*Log Out of an Instance*
`XipcMaskTraps()`	*Activate Trap Mask*
`XipcPing()`	*Detect Remote Host*
`XipcSetOpt()`	*Change Parameter Defaults*
`XipcStart()`	*Start an Instance*
`XipcStop()`	*Stop an Instance*
`XipcSystemErrorReport()`	*Get Error Report*
`XipcTerm()`	*Terminate XIPC Platform Environment*
`XipcUnfreeze()`	*Unfreeze an Instance*
`XipcUnmaskTraps()`	*Deactivate Trap Mask*

X•IPC Macro List

`XIPC_TRAP_FUNCTION_TEST()`	Trap Service Function Test

2. X•IPC COMMANDS

2.1 Platform Commands

2.1.1 XIPCINIT - INITIATE THE X•IPC PLATFORM ENVIRONMENT

NAME

xipcinit - Initiate the X•IPC Platform Environment

SYNTAX

```
xipcinit
```

PARAMETERS

None

RETURNS

Value	Description
RC == 0	Environment initiated successfully.
RC != 0	Error.

DESCRIPTION

This command is used for initializing the X•IPC environment on a computer platform. It should be the first X•IPC command issued on the platform when the platform is started. `xipcinit` sets up all internal structures and background processes needed for supporting X•IPC activity on the platform.

`xipcinit` references the `xipc.env` file located within the `XIPCROOT` location on the platform's file system when it is invoked. The `xipc.env` file provides the list of *program* and *catalog* parameters that are to be set up by `xipcinit`. The overall syntax for `xipc.env,` within a TCP/IP setting, follows, with sections defining *program* parameters preceding those that define *catalog* parameters.

Program Parameters:

```
[XIPCINIT]
START           P1,P2,…     # Start the listed programs at xipcinit time.
                            # Programs started by default if the START parameter
                            # is not present, are: xipcisd, xipclad, xipciad,
                            # xipcicd and xipcidld.

XIPCPATH                    # When specified, XIPCPATH identifies the XIPC
                            # installation directory.  This is typically set
                            # when the installation directory is different from
                            # the XIPCROOT directory.
```

```
# The following parameter sections are used by each
# of the respective platform environment programs when
# they start.  If the parameter section for a particular
# program is not specified, or if the section is specified
# empty, then the program employs its default parameter.
# It is advised that the default values be maintained.

[XIPCICD]
…any xipcicd parameters… # These are described below.

[XIPCISD]
…any xipcisd parameters…     # These are described below.

[XIPCIAD]
…any xipciad parameters…     # These are described below.

[XIPCLAD]
…no parameters…

[XIPCIDLD]
…any xipcidld parameters…    # These are described below.

[XIPCREG]
…any xipcreg parameters…     # These are described below.

[XIPCDREG]
…any xipcdreg parameters…    # These are described below.
```

Catalog Parameters:

```
[CATALOG]
# general catalog parameters
. . .
. . .

[CATALOG.TCPIP]
# TCP/IP specific catalog parameters
. . .
. . .
```

Tables defining the platform environment parameters follow. Refer to the relevant Platform Notes for platform-specific aspects of the programs' operations.

2.1.1.1 Program Parameters

The tables below list the program parameters. Each parameter is presented with its name, description and default value. The order that parameters appear within the respective sections of the configuration is not significant.

XIPCINIT

Parameter Name	Description	Default Value
[XIPCINIT]	xipcinit section header.	- N/A -
START	List of programs to start at xipcinit time. Programs started by default are: xipcicd, xipclad, xipciad, xipcisd and xipcidld. If the START parameter is omitted, the default programs are started. If no daemons are desired, the START parameter should be specified with ***no*** values. In most cases, the START parameter should be omitted. Refer to the parameter tables that follow.	See description at left.
XIPCPATH	When specified, XIPCPATH identifies the *X•IPC* installation directory, allowing installation of the product in a read-only area of the file system. See section 3.3.1 of the *X•IPC* User Manual for further discussion.	XIPCROOT

XIPCICD

This program is started by xipcinit by default.

Parameter Name	Description	Default Value
[XIPCICD]	xipcicd section header.	- N/A -
RECVTIMEOUT	Network receive timeout value (in milliseconds)	[none]
PINGTIMEOUT	Network ping timeout value (in milliseconds)	[none]
PINGRETRIES	Number of network ping retries	3
CONNECTTIMEOUT	Network connect timeout (in seconds)	0

XIPCISD

This program is started by xipcinit by default.

Parameter Name	Description	Default Value
[XIPCISD]	xipcisd section header.	- N/A -
RECVTIMEOUT	Network receive timeout value (in milliseconds)	[none]
PINGTIMEOUT	Network ping timeout value (in milliseconds)	[none]
PINGRETRIES	Number of network ping retries	3
XIPCCAT	As defined in the *X•IPC* User Guide.	Null

Parameter Name	Description	Default Value
XIPCCATLIST	As defined in the X•IPC User Guide.	Null

XIPCIAD

This program is started by `xipcinit` by default.

Parameter Name	Description	Default Value
`[XIPCIAD]`	`xipciad` section header.	- N/A -
`MAX_IN_SESSIONS`	The maximum number of inbound TCP/IP sessions kept open by xipciad in support of incoming asynchronous event traffic.	5
`MAX_OUT_SESSIONS`	The maximum number of outbound TCP/IP sessions that are kept open by xipciad in support of outgoing asynchronous event traffic.	5

XIPCIDLD

This program is started by `xipcinit` by default.

Parameter Name	Description	Default Value
`[XIPCIDLD]`	`xipcidld` section header.	- N/A -
`INTERVAL`	The time to pause between idle user detection cycles, with **s** specifying the interval in seconds, **m** specifying minutes and **h** specifying hours.	30m
`LOGFILENAME`	A file to be used for logging `xipcidld` activity.	[no logging]
`ACTION`	Action to be taken when idle user is detected. Refer to the Idle User presentation in the Technical Notes Appendix of the *X•IPC* User Guide for a list of possible values.	`ABORT`

XIPCREG

XIPCREG is an internal command used by `xipcinit` for registering items. Do not list it as a `START` parameter value.

Parameter Name	Description	Default Value
`[XIPCREG]`	`xipcreg` section header.	- N/A -
`RECVTIMEOUT`	Network receive timeout value (in milliseconds)	[none]
`PINGTIMEOUT`	Network ping timeout value (in milliseconds)	[none]
`PINGRETRIES`	Number of network ping retries	3

XIPCDREG

XIPCDREG is an internal command used by `xipcinit` for deregistering items. Do not list it as a `START` parameter value.

Parameter Name	Description	Default Value
[XIPCDREG]	`xipcdreg` section header.	- N/A -
RECVTIMEOUT	Network receive timeout value (in milliseconds)	[none]
PINGTIMEOUT	Network ping timeout value (in milliseconds)	[none]
PINGRETRIES	Number of network ping retries	3

2.1.1.2 General Catalog Parameters

The table below lists the general catalog configuration parameters, each presented with its name, description and default value. The order of parameters within the `[CATALOG]` section of the configuration is not significant.

Parameter Name	Description	Default Value
`[CATALOG]`	Catalog section header.	- N/A -
`MAX_NAMESPACES`	Maximum number of namespaces that can be defined within the catalog.	8
`MAX_NODES`	Maximum number of network nodes that can be registered within the catalog.	31
`MAX_INSTANCES`	Maximum number of instances that can be registered within the catalog.	31
`MAX_APPQUEUES`	Maximum number of app-queues that can be registered within the catalog.	128

2.1.1.3 Protocol-Specific Catalog Parameters

The table below lists the TCP/IP protocol-specific catalog configuration parameters. Each parameter is presented with its name, description and a default value, where relevant.

Parameter Name	Description	Default Value
`[CATALOG.TCPIP]`	Catalog protocol header for TCP/IP	- N/A -
`NAMESPACE`	Defines an *X•IPC* namespace.	[none]

2.1.2 XIPCTERM - TERMINATE THE X-IPC PLATFORM ENVIRONMENT

NAME

xipcterm - Terminate the X-IPC Platform Environment

SYNTAX

```
xipcterm
```

PARAMETERS

None .

RETURNS

Value	Description
RC == 0	X-IPC terminated successfully.
RC != 0	Error.

DESCRIPTION

This command is used to terminate the X-IPC environment on a computer platform. It should be the last X-IPC command issued on the platform when the platform is stopped. **xipcterm** closes all internal structures and background processes needed for supporting X-IPC activity on the platform.

Refer to the relevant Platform Notes for platform-specific aspects of **xipcterm**.

2.2 Instance Commands

2.2.1 THE INSTANCE ENVIRONMENT

X•IPC instances can be established in network, local and stand-alone environments; these are described below. The environment variables are then defined in the following section.

2.2.1.1 X•IPC Network Environment

An X•IPC network instance is an X•IPC instance that is named and whose name is registered within the X•IPC namespace catalog environment.

An X•IPC network instance is ideal for establishing an instance that:

- must be accessed in a network-transparent manner across the network
- is used, therefore, to advantage by QueSys, SemSys and MemSys programming, where network-transparent access to an instance is a primary feature

An X•IPC network instance may be accessed via its network name (locally or remotely), by its Instance File Name or via the `NETNAME` parameter in the [`XIPC`] section of the Instance Configuration File.

The following environment variables are used by an X•IPC network instance:

- `XIPCROOT`
- `XIPC`
- `XIPCCAT`
- `XIPCCATLIST`
- `XIPCHOST`
- `XIPCHOSTLIST`
- `XIPCASYNCIO`

2.2.1.2 X•IPC Local Environment

An X•IPC local instance is an X•IPC instance that is named, but whose name is only visible within the bounds of the node on which it is started.

An X•IPC local instance is ideal for establishing an X•IPC instance that is:

- inaccessible from any remote node
- accessible within its platform in an operating system transparent manner (i.e., by its name)
- used to advantage by MomSys programming because that environment most often invokes processes logging into instances on the local node.

Such an instance may be accessed either by its local name *(@InstanceName)*, by its Instance File Name or via the `LOCALNAME` parameter in the [`XIPC`] section of the Instance Configuration File.

The following environment variables are used by an X•IPC local instance:

- `XIPCROOT`
- `XIPC`
- `XIPCASYNCIO`

2.2.1.3 X•IPC Stand-Alone Environment

An X•IPC stand-alone instance is an X•IPC instance that is *not* named or registered in any manner within any X•IPC naming catalog. As such, it is ideal for establishing an X•IPC instance that is:

- inaccessible from any remote node
- invisible (except to programs that use it) within the node on which it is running
- used by intra-nodal X•IPC applications where no networking is involved

Because it is unnamed, an X•IPC stand-alone instance is sometimes referred to as an *anonymous* instance.

The only way to access an X•IPC stand-alone instance is via its Instance File Name. X•IPC stand-alone instances have a distinct advantage over local and network instances in that they can be embedded within an application and made practically invisible on the platform on which they are running. This is because they have no name that is registered in any X•IPC server.

A disadvantage of an X•IPC stand-alone instance is that its only form of access is via its Instance File Name. This means that code accessing the stand-alone instance will be platform-specific, because file naming conventions differ from platform to platform.

The following environment variables are used by an X•IPC stand-alone instance:

- `XIPCROOT`
- `XIPC`
- `XIPCASYNCIO`

2.2.2 THE ENVIRONMENT VARIABLES

For an important description of the X•IPC Platform Environment and Environment Commands/Variables, please read chapter 3 of the X•IPC User Guide.

XIPCROOT - Platform Directory Environment Variable (Network, Local and Stand-alone)

NAME

XIPCROOT - Platform Directory Environment Variable

DESCRIPTION

XIPCROOT is the environment variable used to designate the platform directory in which X•IPC was installed**. It is required in all cases.**

COMMANDS REFERENCING XIPCROOT

All X•IPC programs and utilities

FUNCTIONS REFERENCING XIPCROOT

None.

XIPC - Instance Name Environment Variable (Network, Local and Stand-alone)

NAME

XIPC - Instance Name Environment Variable

DESCRIPTION

XIPC is one of the environment variables used in conjunction with the *X•IPC* / network environment. When XIPC is set, it is assumed to contain an Instance Network Name (@[node:]NetworkName) or an Instance File Name as its value.

X•IPC commands requiring specification of an instance name as an argument use the XIPC environment variable if the argument is omitted.

COMMANDS REFERENCING **XIPC**

xipcstart, xipcstop, momview, queview, semview, memview, xipc

FUNCTIONS REFERENCING **XIPC**

None.

XIPCASYNCIO – Asynchronous I/O Descriptor Environment Variable (Network, Local and Stand-alone)

NAME

XIPCASYNCIO - The Asynchronous I/O Descriptor Environment Variable

---

DESCRIPTION

Setting the `XIPCASYNCIO` environment variable to any non-`NULL` value directs *X•IPC* to establish the process's *X•IPC* asynchronous notification mechanism to use an I/O descriptor instead of a signal.

The environment variable must be set at the time that the process issues an `XipcLogin()` function call, in order for the environment variable to have its effect. Otherwise, the default (i.e. signal) mechanism is set up.

See the Technical Note "Using I/O Descriptors for Asynchronous Operations on UNIX" in the Appendix to the *X•IPC* User Manual.

COMMANDS REFERENCING **XIPCASYNCIO**

None

FUNCTIONS REFERENCING **XIPCASYNCIO**

`XipcLogin(), XipcAsyncIoDescriptor(), XipcAsyncEventHandler()`

XIPCCAT - Catalog Node Environment Variable (Network only)

NAME

XIPCCAT - Catalog Node Environment Variable

DESCRIPTION

XIPCCAT is an environment variable that specifies a list of one or more *node* names as a string, with the nodes separated by white space, commas, colons or semi-colons.

The nodes specified are *Instance Catalog* nodes that maintain a catalog of registered network instances and are used to locate *X•IPC* instances during login.

The following list describes the order in which instance search range specification parameters are used:

1. The environment variable XIPCHOST.
2. The environment variable XIPCHOSTLIST.
3. The environment variable XIPCCAT.
4. The environment variable XIPCCATLIST.

When more than one search specification is present, X•IPC uses the first one in the order listed above and ignores the rest. (The other environment variables are presented elsewhere in this section.) Thus, if the XIPCHOST or XIPCHOSTLIST environment variables are set, then the XIPCCAT environment variable is ignored. If XIPCCAT is set, then the XIPCCATLIST environment variable is ignored.

COMMANDS REFERENCING **XIPCCAT**

xipcstart, xipcstop

FUNCTIONS REFERENCING **XIPCCAT**

XipcStart(), XipcStop(), XipcLogin()

XIPCCATLIST - Catalog Node List Environment Variable (Network only)

NAME

XIPCCATLIST - Catalog Node List Environment Variable

DESCRIPTION

XIPCCATLIST is an environment variable that specifies a name of a *file* containing a list of nodes, one per line.

The nodes specified are *Instance Catalog* nodes that maintain a catalog of registered network instances and are used to locate X•IPC instances during login.

The following list describes the order in which instance search range specification parameters are used:

1. The environment variable XIPCHOST.
2. The environment variable XIPCHOSTLIST.
3. The environment variable XIPCCAT.
4. The environment variable XIPCCATLIST.

When more than one search specification is present, X•IPC uses the first one in the order listed above and ignores the rest. (The other environment variables are presented elsewhere in this section.) Thus, if XIPCHOST, XIPCHOSTLIST or XIPCCAT are set, then the XIPCCATLIST environment variable is ignored.

COMMANDS REFERENCING XIPCCATLIST

xipcstart, xipcstop

FUNCTIONS REFERENCING XIPCCATLIST

XipcStart(), XipcStop(), XipcLogin()

XIPCHOST - Instance Node Environment Variable (Network only)

NAME

XIPCHOST - Instance Node Environment Variable

DESCRIPTION

When `XIPCHOST` is set, it is assumed to contain a list of one or more node names, separated by white space, commas, colons or semi-colons, wherein which instance searches are to be confined.

The following list describes the order in which instance search range specification parameters are used:

1. The environment variable `XIPCHOST.`
2. The environment variable `XIPCHOSTLIST.`
3. The environment variable `XIPCCAT.`
4. The environment variable `XIPCCATLIST.`

When more than one search specification is present, X•IPC uses the first one in the order listed above and ignores the rest. (The other environment variables are presented elsewhere in this section.) Thus, if `XIPCHOST` is set, then the `XIPCHOSTLIST, XIPCCAT` and `XIPCCATLIST` environment variables are ignored.

COMMANDS REFERENCING **XIPCHOST**

`xipcstart`

FUNCTIONS REFERENCING **XIPCHOST**

`XipcStart(), XipcLogin()`

XIPCHOSTLIST - Instance Node List Environment Variable (Network only)

NAME

XIPCHOSTLIST - Instance Node List Environment Variable

DESCRIPTION

When XIPCHOSTLIST is set it is assumed to identify the name of a *file* containing a list of network node names (one per line) wherein which all instance searches are to occur. Instance searches will then be limited to the listed network nodes. XIPCHOSTLIST, as such, can be used to define a multiple node instance search range.

The following list describes the order in which instance search range specification parameters are used:

1. The environment variable XIPCHOST.
2. The environment variable XIPCHOSTLIST.
3. The environment variable XIPCCAT.
4. The environment variable XIPCCATLIST.

When more than one search specification is present, X•IPC uses the first one in the order listed above and ignores the rest. (The other environment variables are presented elsewhere in this section.) Thus, if XIPCHOST is set, then the XIPCHOSTLIST environment variable is ignored; the search range is then limited to the nodes identified by the value of XIPCHOST. If XIPCHOSTLIST is set, then the XIPCCAT and XIPCCATLIST environment variables are ignored.

COMMANDS REFERENCING XIPCHOSTLIST

xipcstart

FUNCTIONS REFERENCING XIPCHOSTLIST

XipcStart(), XipcLogin()

2.2.3 XIPCSTART - START AN X•IPC INSTANCE

NAME

xipcstart - Start an X•IPC Instance

SYNTAX

xipcstart [*InstFileName*] [-l*InstLocalName*|-n*InstNetName*] [-i] [-t]

PARAMETERS

Name	Description
InstFileName	The instance configuration file name of the instance to be started (i.e., the path name of its instance configuration file). This is referred to as an Instance File Name.
-l*InstLocalName*	The Instance Local Name assigned to the instance. The *-l* flag and the *-n* flag are mutually exclusive.
-n*InstNetName*	The Instance Network Name to be assigned to the instance. The *-l* flag and the *–n* flag are mutually exclusive.
-i	Initialization flag to create a new MomSys instance with no history (i.e., starting with a clean message repository).
-t	Test flag to generate an instance configuration report without starting the instance.

RETURNS

Value	Description
RC == 0	Instance started successfully.
RC != 0	Instance not started.

DESCRIPTION

This program is used to start and initialize an instance of X•IPC. It must be executed before any program can log into that instance. *Note that, before* xipcstart *or any other command can be invoked,* XIPCROOT *(See section 2.2.2) must be set to the path of the Platform Directory in which X•IPC was installed.*

The specified *InstFileName* is the path name of the instance configuration file.

If *InstFileName* is not specified, the value of the XIPC environment variable is used as the instance file name of the instance to be started. Multiple instances can be started on a single platform, provided that each one uses a unique *InstFileName*.

An instance that is to be started for use in a *local* environment must be started with the "`-l`*InstLocalName*" flag (or with the `LOCALNAME` parameter in the Instance Configuration File). The instance can then be accessed by processes within its platform in an operating system transparent manner via the instance's local name.

If a local instance, having the specified local name, already exists on that platform–within the user's instance search range–then `xipcstart` fails.

An instance that is to be started for use in a *network* environment must be started with the "`-n`*InstNetName*" flag flag (or with the `NETNAME` parameter in the Instance Configuration File). The instance can then be accessed by processes across the network via the instance's network name.

If an Instance Name (local or network) is not specified in the command syntax, the name will be taken from the parameter specified in the `[XIPC]` section of the Instance Configuration File. (See Section 2.2.4.)

If no naming parameters are specified within the `.cfg` file either, then the instance is started as a "stand-alone" instance having no registered name. Such an instance is only accessible via its Instance File Name. Refer to `XipcLogin()` for further information.

If a network instance having the specified network name already exists on the network–within the user's instance search range–then `xipcstart` fails. The instance search range is a function of the `XIPCHOST, XIPCHOSTLIST, XIPCCAT` and `XIPCCATLIST` environment variables. For details of *X•IPC /*Network environment variables, refer to the previous section.

Note that *X•IPC* instances that are started with an assigned name (either a *Local* or a *Network* name) are visible to the `xipclist` utility command. It is sometimes desirable that an instance's existence not be visible to `xipclist`. This can be accomplished by assigning the instance a name starting with the '_' (underscore) character. So for example: an instance named `foo` would be visible to `xipclist`, while an instance named `_foo` would not.

Note that command parameters override parameters specified in the Instance Configuration File. The expected format and contents of a configuration file are detailed below.

- If "-i" (the *InitializationFlag*) is specified, the database is reinitialized. **All data (local app-queues, remote app-queues and messages) will be destroyed.** Use this option *only* if there is a need to restart an instance with a fresh message repository. The *InitializationFlag* applies only to the disk-based MomSys subsystem.
- If "-t" (the *TestFlag*) is specified, the command generates an instance configuration report and does *not* start the instance. This can be used to determine the memory requirements of a particular configuration file without having all required resources available.

ERRORS

Display messages.

2.2.4 INSTANCE CONFIGURATION FILE FORMAT

NAME

Instance Configuration Files - X•IPC Instance Configuration Files

SYNTAX

```
[XIPC]                  /* defined below */
ParmName  ParmValue
ParmName  ParmValue
ParmName  ParmValue

[MOMSYS]                /* defined in the MomSys documentation */
ParmName  ParmValue
ParmName  ParmValue
...

[QUESYS]                /* defined in the QueSys/SemSys/MemSys
ParmName  ParmValue        documentation */
ParmName  ParmValue
...

[SEMSYS]                /* defined in the QueSys/SemSys/MemSys
ParmName  ParmValue        documentation */
ParmName  ParmValue
...

[MEMSYS]                /* defined in the QueSys/SemSys/MemSys
ParmName  ParmValue        documentation */
ParmName  ParmValue
...

[IDLE_USER]     /* defined in "XIPC Idle User Detection
Mechanism" Technical Note Appendix  */
ParmName  ParmValue

[REMOTE_USER]              /* defined below */
ParmName  ParmValue
```

PARAMETERS

Name	Description
ParmName	The name of a configuration parameter.

ParmValue The value of the configuration parameter.

DESCRIPTION

An X•IPC instance configuration file completely describes an X•IPC instance. It contains all information needed to parameterize the instance.

The [XIPC] section supports the following parameters:

Parameter	Value	Default
SHARED_MEM	Single/Multiple	Multiple If Single is specified, see the relevant Platform Notes for platform-specific details.
NAMESPACE	The name of the X•IPC namespace with which to affiliate the instance, or	There is *no* default. See the MomSys User Guide for information on the X•IPC namespace. Note that this parameter is currently not used for QueSys, SemSys or MemSys.
CSEC_ALGORITHM	Gate/Semaphore where: The Semaphore algorithm should be selected for multiprocessor environments.	Gate (normal algorithm)
NETNAME	Instance Network Name assigned to the instance. The NETNAME and LOCALNAME parameters are mutually exclusive.	None
LOCALNAME	Instance Local Name assigned to the instance. The NETNAME and LOCALNAME parameters are mutually exclusive.	None

The [REMOTE_USER] section (formerly known as a [NETWORK] section) is required if the instance is to be started as a network instance. It supports the following parameter:

Parameter	Value	Default
MAX_TEXTSIZE	The maximum size (in bytes) of an instance's staging area into which a message is sent to/from a client program.	1024 To override the default, a user program can call XipcSetOpt() *before login* to establish a different value.

The order of the contained sections is insignificant. Blank lines are ignored. Comment lines start with any non-alphanumeric character. Comments can also follow *ParmValue* on the same line.

Details regarding configuration file format for the MomSys subsystem can be found in the MomSys Reference Manual; for the QueSys, MemSys and SemSys subsystems, configuration file format information can be found in the QueSys/MemSys/SemSys User Guide.

Certain operating system platforms introduce a number of platform-specific instance configuration parameters. Refer to the respective Platform Notes for details.

A significant change from X•IPC Version 2.8 is that, if an instance is not intended to support the operations of one or more subsystems, the [SectionName] should simply be *omitted* from the configuration file; inclusion of the [SectionName] without specifying parameter values will result in the default values being used (in contrast

to Version 2.8, in which specifying the `[SectionName]` without parameters served to define a null subsystem with respect to the instance).

2.2.5 XIPCLIST - LIST ACTIVE LOCAL AND NETWORK INSTANCES

NAME

xipclist - List Active Local and Network Instances

SYNTAX

xipclist [*NodeNameList*]

PARAMETERS

Name	Description
NodeNameList	The machine name or names to which **xipclist**'s reporting should be limited. No node, one node or a list of nodes can be designated.

Examples:

```
xipclist

xipclist NodeA

xipclist NodeA,NodeB,NodeC
```

RETURNS

Value	Description
RC == 0	List produced successfully.
RC != 0	Error.

DESCRIPTION

If *NodeNameList* is specified, **xipclist** reports the network instances on the specified node(s), and no others. In such a case, the instance search range defined by the environment variables is ignored.

If *NodeNameList* is *not* specified, this program lists all network instances within the search range defined by the XIPCHOST, XIPCHOSTLIST, XIPCCAT and XIPCCATLIST environment variables.

The following list describes the order in which instance search range specification parameters are used:

1. The environment variable XIPCHOST.
2. The environment variable XIPCHOSTLIST.
3. The environment variable XIPCCAT.
4. The environment variable XIPCCATLIST.

When more than one search specification is present, X•IPC uses the first one in the order listed above and ignores the rest. (The environment variables were presented earlier.)

It is sometimes desirable that an instance's existence not be visible to xipclist. An X•IPC instance that was started with an assigned name (either a *Local* or a *Network* name) will not visible to the xipclist utility command if its name starts with the '_' (underscore) character, e.g., _foo.

ERRORS

Display messages.

2.2.6 XIPCSTOP - STOP AN X•IPC INSTANCE

NAME

xipcstop - Stop an X•IPC Instance

SYNTAX

```
xipcstop [InstFileName|InstanceName] [-f]
```

PARAMETERS

Name	Description
InstFileName	The instance configuration file name (i.e., the path name of its instance configuration file) which is to be stopped.
InstName	The name of the instance which is to be stopped as specified when started via `xipcstart`, beginning with the character '@'. The Instance name can be omitted, in which case the value of the `XIPC` environment variable is used. Example: `xipcstop @xyz`
-f	'Force' flag to forcefully clean system resosurces held by the specified instance.

RETURNS

Value	Description
RC == 0	Instance stopped successfully.
RC != 0	Instance not stopped.

DESCRIPTION

This program is used to shut down and terminate an instance of X•IPC. All resources–other than disk-based MomSys app-queues–held by that instance are released. Users of the instance which have not logged out are forcibly aborted from the instance. Disk-based MomSys app-queues are preserved on disk until the next time the instance is started.

The specified *InstFileName* or *InstName* identifies the instance to be brought down. If *InstFileName* or *InstName* is not specified, the value of the `XIPC` environment variable is used as the instance file name of the instance to be stopped.

The `[-f]` Force flag forcefully cleans system resources held by the specified instance. WARNING: This flag should *only* be employed when a standard `xipcstop` command "crashes."

ERRORS

Display messages.

2.3 xipc - X•IPC Interactive Command Processor

NAME

xipc - X•IPC Interactive Command Processor

SYNTAX

```
xipc
```

PARAMETERS

None

RETURNS

Not Applicable

DESCRIPTION

`xipc` is a command interpreter that provides the user with interactive access to X•IPC API capabilities.

Most of the interpreter's commands correspond to X•IPC API's, and their arguments are the same, except for necessary adjustments to the interactive environment. To find a full description of a command and its arguments, refer to the description of the corresponding API.

2.3.1 THE X•IPC INTERACTIVE LANGUAGE

2.3.1.1 SYNTAX

Each command starts with a command verb, usually an X•IPC API name. The command name is followed by the command arguments separated by one or more spaces.

Arguments that consist of a list of values, such as *SidList*, use a comma as a separator between the values.

Text arguments are entered either as a string of characters delimited by spaces or as a string delimited by double quotes. When quotes are used as delimiters, a quote character can also be specified as part of the string by preceding it with a back-slash (\) character.

A line starting with the character "#" is treated as a comment line and its contents are ignored.

2.3.1.2 Variables

`xipc` provides four sets of built-in variables:

- ACB's - `xipc` defines 26 ACB variables identified by the letters `a` through `z`.
- Message Headers - `xipc` defines 26 message header variables identified by the letters `a` through `z`.
- Memory Sections - `xipc` defines 26 memory section variables identified by the letters `a` through `z`.

- MomSys Message Ids - `xipc` defines 26 message id variables identified by the letters `a` through `z.`

2.3.1.3 Callback Routines

`xipc` has two groups of callback routines that can be used in conjunction with asynchronous operations:

- Six callback routines named `cb1` through `cb6` that display the results of the completing operation.
- Twenty-six callback routines named `cba` through `cbz`. Each of these callback routines can be assigned an `xipc` command to execute when the asynchronous operation completes.

2.3.1.4 Blocking Options

Many of `xipc`'s commands have a blocking option parameter. This parameter corresponds to the blocking option of X•IPC API's. The syntax of the blocking option is one of the following:

- **`wait`**
- **`nowait`**
- **`timeout`**(*`Seconds`*)

  `Seconds` - Timeout length in seconds.

- **`callback`**(*`CallbackAction`*, *`AcbId`*)

  `CallbackAction` - Either a name of a predefined callback routine (`cb1-cb6` or `cba-cbz`) or an xipc command enclosed in double quotes to be executed when the asynchronous operation completes.

  `AcbId` - ACB variable (a-z).

- **`post`**(*`Sid`*, *`AcbId`*)

  `Sid` - Semaphore Id to be set when the operation completes.

  `AcbId` - ACB variable (a-z).

- **`ignore`**(*`AcbId`*)

  `AcbId` - ACB variable (a-z).

Note that all flags must be specified *before* (to the left of) the blocking option.

2.3.1.5 Conventions Used In This Section

The following conventions are used in the description of `xipc` command syntax:

- Text in **`bold`** is to be entered as specified;
- Items in *`italics`* represents values to be provided by the user;
- Items between brackets [ ] designate an optional choice.

- Items between braces{} designate a mandatory choice.

2.3.2 GENERAL INTERACTIVE COMMANDS

2.3.2.1 ! - Execute Operating System Command

SYNTAX

! *Command*

ARGUMENTS

Command Native operating system command.

EXAMPLES

```
xipc> # Unix example of operating system command
xipc> !date
Thu May 21 10:58:20 EDT 2003

xipc> # VMS example of operating system command
xipc> !show time
21-May-2003 10:58:20

xipc> # Windows example of operating system command
xipc> !date
The current date is: Thu 5-21-2003
Enter the new date: (mm-dd-yy)
```

2.3.2.2 acb - Display Contents of ACB

SYNTAX

acb *AcbId*

ARGUMENTS

AcbId One letter identification of the ACB.

EXAMPLES

```
xipc> acb a
      AUid            = 33
      AsyncStatus     = XIPC_ASYNC_INPROGRESS
      UserData1       = 000000000
      .
      .
      .
```

2.3.2.3 callback - Assign Callback Command

SYNTAX

callback *CallbackName* *XipcCommand*

ARGUMENTS

CallbackName The name of a callback routine (`cba-cbz`).

XipcCommand An `xipc` command enclosed in double quotes.

EXAMPLES

```
xipc> # Start spooling when queue fills up
xipc> callback cba "quespool 2 /usr/tmp/spl"
      Command saved
```

2.3.2.4 help - Display List Of Arguments

SYNTAX

help *Command*
? *Command*

ARGUMENTS

Command Name of `xipc` command.

EXAMPLES

```
xipc> help xipclogin
      xipclogin
         InstanceName
         UserName
```

2.3.2.5 quit - Logout And Quit

SYNTAX

q[uit]

ARGUMENTS

None.

EXAMPLES

```
xipc> q
      Logging out user 11 from: @Server
      Logging out user 31 from: @DBServer
```

2.3.2.6 uid - Display Current User Id

SYNTAX

uid

ARGUMENTS

None.

EXAMPLES

```
xipc> uid
      Uid = 11
```

2.3.3 X-IPC INTERACTIVE COMMANDS

2.3.3.1 xipcabort - Abort a User

SYNTAX

xipcabort *UserId*

ARGUMENTS

UserId	User id of user to be aborted

EXAMPLES

```
xipc> xipcabort 11
      RetCode = 0
```

2.3.3.2 xipcconnect - Connect to a Login

SYNTAX

xipcconnect *[InstanceName] [UserId]*

ARGUMENTS

InstanceName	Name of instance to connect to: Either an instance configuration file name or an instance name (local or network) starting with the character '@'. Instance name can be specified as '*' in which case the value of the environment variable `XIPC` will be used. The instance name must be specified exactly as it was specified in the xipclogin command.
UserId	User id as returned by `xipclogin`

EXAMPLES

```
xipc> # Log into stand-alone instance.
      # Disconnect from the login.
      # Then reconnect to the login.
xipc> xipclogin /usr/xipc/test Joe
      Uid = 11
xipc> xipcdisconnect
      RetCode = 0
xipc> xipcconnect /usr/xipc/test 11
      RetCode=0
```

2.3.3.3 xipcdisconnect - Disconnect from a Login

SYNTAX

xipcdisconnect

ARGUMENTS

None

EXAMPLES

```
xipc> xipclogin /usr/xipc/test Joe
      Uid = 11
xipc> xipcdisconnect
      RetCode = 0
xipc> xipclogin /usr/xipc/test2 Joe
      Uid = 7
```

2.3.3.4 xipcerror - Display Error Message

SYNTAX

xipcerror *ErrorCode*

ARGUMENTS

ErrorCode X•IPC error code

EXAMPLES

```
xipc> xipcerror -1003
      Configuration capacity limit exceeded
```

2.3.3.5 xipcfreeze - Freeze Instance

SYNTAX

xipcfreeze

ARGUMENTS

None.

EXAMPLES

```
xipc> xipcfreeze
      RetCode = 0
```

2.3.3.6 xipcgetopt – Get Parameters

SYNTAX

xipcgetopt [*Option*]

ARGUMENTS

[*Option*]	One from the following options: CONNECTTIMEOUT, RECVTIMEOUT, PINGTIMEOUT, PINGRETRIES, PINGFUNCTION, PRIVATEQUEUE, MAXTEXTSIZE, ASYNCFD

EXAMPLES

```
xipc> xipcgetopt pingtimeout
      Parameter [pingtimeout] –> : [5]
```

2.3.3.7 xipcidlewatch - Control Idle Watch Monitoring

SYNTAX

Xipcidlewatch *[Option]*

ARGUMENTS

Option	One of “start,” “stop” or “mark.”

EXAMPLES

```
xipc> xipcidlewatch start
      RetCode = 0
```

2.3.3.8 xipcinfologin - Get Login Information

SYNTAX

xipcinfologin

ARGUMENTS

None

EXAMPLES

```
xipc> xipcinfologin
      Uid    Instance
      ---    --------
      11    /usr/xipc/test
       7    @Server
      31    @DBServer
```

2.3.3.9 xipcinfoversion - Get XIPC Version Information

SYNTAX

xipcinfoversion | xipcver

ARGUMENTS

None

EXAMPLES

```
xipc>  xipcinfoversion
       XIPC Version 3.4.0 aa - Windows NT 4.0
```

2.3.3.10 xipcinit – Initiate Platform Environment

SYNTAX

xipcinit

ARGUMENTS

None

EXAMPLES

```
xipc> xipcinit
      xipcinit: XIPC Platform Environment Initiated
      Win32 – XIPC 3.4.0aa [Build 5012]
      RetCode = 0
```

2.3.3.11 xipclist - List Active Network Instances

SYNTAX

xipclist *[NodeName]*

ARGUMENTS

NodeName Name of node about which xipclist's reporting should be limited.

EXAMPLES

```
xipc> xipclist
      Machine.................[grumpy]
      Instance Name...........[server]
      Instance File Name......[/xipc/server]
      Maximum Text Size.......[1024]
```

2.3.3.12 xipclogin - Log Into An Instance

SYNTAX

xipclogin *[InstanceName] [UserName]*

ARGUMENTS

InstanceName	Name of instance to log into: Either an Instance File Name or an instance name (local or network) starting with the character '@'. Instance name can be specified as '*' in which case the value of the environment variable XIPC will be used.
UserName	Name to be assigned to the X•IPC user. If omitted, the string XIPC will be used. If the name "superuser" is used, the user is logged in as a superuser.

EXAMPLES

```
xipc> xipclogin  /tmp/config George
      Uid = 11

xipc> # Log into instance “Server”. “xipc” is default user name
xipc> xipclogin  @Server
      Uid = 1

xipc> # Log into network instance on node “dopey”
xipc> xipclogin  @dopey:Server George
      Uid = 1
```

2.3.3.13 xipclogout - Log Out Of Instance

SYNTAX

xipclogout

ARGUMENTS

None.

EXAMPLES

```
xipc> xipclogout
      RetCode = 0
```

2.3.3.14 xipcmasktraps - Activate Trap Mask

SYNTAX

xipcmasktraps

ARGUMENTS

None.

EXAMPLES

```
xipc> xipcmasktraps
      RetCode = 0
```

2.3.3.15 xipcsetopt - Set Parameters

SYNTAX

xipcsetopt [*Option*]

ARGUMENTS

[*Option*]	One from the following options: CONNECTTIMEOUT, RECVTIMEOUT, PINGTIMEOUT, PINGRETRIES, PINGFUNCTION, PRIVATEQUEUE, MAXTEXTSIZE, ASYNCFD

EXAMPLES

```
xipc> xipcsetopt pingtimeout 5
      Parameter [pingtimeout] –> New value [5]
```

2.3.3.16 xipcstart - Start An Instance

SYNTAX

xipcstart *InstFileName InstName [Options]*

ARGUMENTS

InstFileName — The instance configuration file name of instance to be started (i.e., the path name of its instance configuration file). The Instance File Name can be omitted, in which case the value of the environment variable `XIPC` will be used.

InstName — Name to be assigned to the instance. The Instance Name can be omitted, in which case the optional value (`LOCALNAME` or `NETNAME`) in the [`XIPC`] section of the Instance Configuration File may be used. Note that, should the Instance Name be omitted, and if `local` is not specified, `network` is the default (as shown in the example below). (See *`[Options]`* below for setting an instance as `local` or `network`.)

Note that X•IPC instances that are started with an assigned name (either a *Local* or a *Network* name) are visible to the `xipclist` utility command. It is sometimes desirable that an instance's existence not be visible to `xipclist`. This can be accomplished by assigning the instance a name starting with the '_' (underscore) character. So for example: an instance named `foo` would be visible to `xipclist`, while an instance named `_foo` would not.

[Options] — One or more of the following: `initialize, network, local, report, test` or `0`. When listing multiple options, they are listed and separated by commas.

Note: Asterisks (*) can be used as "place holders," with the defaults noted above, if the arguments preceding *`[Options]`* are not specified. See the example below.

EXAMPLES

```
xipc> # Start a Network instance
xipc> xipcstart  /tmp/config  Server
      XipcStart(SemSys):
      .
      .
      XipcReg: Network Instance [Server] Registered.
      RetCode = 0

xipc> # Start a Stand-Alone Instance.
xipc> # Use XIPC environment variable to specify instance name.
xipc> # Do not output report.
xipc> xipcstart  * * 0
      RetCode = 0
```

2.3.3.17 xipcstop - Stop An Instance

SYNTAX

xipcstop *InstanceName [Options]*

ARGUMENTS

InstanceName	Name of instance to be stopped: Either an Instance File Name or an instance name (local or network) starting with the character '@'. The instance name can be omitted and specified as '*', in which case the value of the environment variable `XIPC` will be used.
Options	One of: `report, force` or `0`.

EXAMPLES

```
xipc> # Use XIPC environment variable to specify instance name.
xipc> # Do not output report.
xipc> xipcstop  * 0
      RetCode = 0
```

2.3.3.18 xipcterm – Terminate Platform Environment

SYNTAX

```
xipcterm
```

ARGUMENTS

None

EXAMPLES

```
xipc> xipcterm
      xipcterm:XIPC Platform Environment Terminated
      RetCode = 0
```

2.3.3.19 xipcunfreeze - Unfreeze Instance

SYNTAX

xipcunfreeze

ARGUMENTS

None.

EXAMPLES

```
xipc> xipcunfreeze
      RetCode = 0
```

2.3.3.20 xipcunmasktraps - Deactivate Trap Mask

SYNTAX

xipcunmasktraps

ARGUMENTS

None.

EXAMPLES

```
xipc> xipcunmasktraps
      RetCode = 0
```

3. X♦IPC FUNCTIONS

3.1 XipcAbort() - Abort a User by Forcing a Log Out

NAME

`XipcAbort()` - Abort a User by Forcing a Log Out

SYNTAX

```
#include "xipc.h"


XINT
XipcAbort(Uid)

XINT Uid;
```

PARAMETERS

Name	Description
Uid	The User ID of the user to be aborted.

RETURNS

Value	Description
RC >= 0	Abort successful.
RC < 0	Error (see error codes below).

DESCRIPTION

XipcAbort() logs the specified user out of an X♦IPC instance. Resources held by the user are released, as follows:

- All semaphores held by the user are released. (On UNIX platforms, the performer of the XipcAbort() must be the same user or another user with `root` privileges.)
- If user *Uid* is currently blocked on a SemAcquire() or SemWait() operation, that operation is cancelled before the user is logged out and the `SEM_ER_NOTLOGGEDIN` error code is returned.
- If the aborted user is currently blocked on a QuePut(), QueGet(), QueWrite(), QueSend() or QueReceive() operation, then that operation is cancelled before the user is logged out and the `QUE_ER_NOTLOGGEDIN` error code is returned.

- All MomSys app-queues that were created by User *Uid* having the `MOM_ATTR_SET_USER_ATTACHED` attribute set are automatically deleted.
- All memory sections owned by the user are released.
- If user *Uid* is currently blocked on a blocking MemSys operation (i.e., MemWrite(), MemRead(), MemLock() or MemSecOwn()), that operation is cancelled before the user is logged out and the `MEM_ER_NOTLOGGEDIN` error code is returned.

A user cannot call XipcAbort() specifying his own *Uid*.

ERRORS

Code	Description
`XIPC_ER_BADUID`	Invalid *Uid* parameter.
`XIPC_ER_NOTLOGGEDIN`	User not logged into X•IPC (User never logged in, is disconnected or was aborted).
`XIPC_ER_SYSERR`	An internal error has occurred while processing the request.
`XIPCNET_ER_CONNECTLOST`	Connection to instance lost.
`XIPCNET_ER_NETERR`	Network transmission error.
`XIPCNET_ER_SYSERR`	Operating system error.

INTERACTIVE COMMAND

SYNTAX

`xipcabort` *`UserId`*

ARGUMENTS

`UserId` User id of user to be aborted

EXAMPLES

```
xipc> xipcabort 11
      RetCode = 0
```

3.2 XipcAsyncEventHandler() – Process Completing *X•IPC* Asynchronous Operations

NAME

`XipcAsyncEventHandler()` - Process Completing *X•IPC* Asynchronous Operations

SYNTAX

```
#include "xipc.h"

XINT
XipcAsyncEventHandler()
```

PARAMETERS

None.

RETURNS

Value	Description
RC >= 0	Success.
RC < 0	Error (see error codes below).

DESCRIPTION

XipcAsyncEventHandler() processes completing asynchronous *X•IPC* operations and reads all data on the *X•IPC* async I/O descriptor. The function should be executed when a process is notified that one of its asynchronous *X•IPC* operations is complete. This generally occurs following the occurrence of a "data ready" event on the *X•IPC* asynchronous I/O descriptor.

The call to XipcAsyncEventHandler() may be placed within the main-line logic, within a signal handler or within an X-Windows event handler.

Note that XipcAsyncEventHandler() blocks if called when there are no outstanding AEBs; therefore, don't call this function until the `select()` call returns, indicating "data ready." Refer to the Technical Note "Using I/O Descriptors for Asynchronous Operations on UNIX" (in the Appendix to the *X•IPC* User Manual) for a program outline.

The XipcAsyncEventHandler() function should only be used when the process has chosen the I/O descriptor method of asynchronous notification by setting the `XIPCASYNCIO` environment variable (described in section 2.2.2).

ERRORS

Code	Description
`XIPC_ER_NOACCESS`	Process not using *X•IPC* asynchronous I/O descriptor method.

XIPC_ER_SYSERR	An internal error has occurred while processing the request.

3.3 XipcAsyncIoDescriptor() – Access the Value of the *X•IPC* Asynchronous I/O Descriptor

NAME

`XipcAsyncIoDescriptor()` - Access the Value of the *X•IPC* Asynchronous I/O Descriptor

SYNTAX

```
#include "xipc.h"


XINT
XipcAsyncIoDescriptor()
```

PARAMETERS

None.

RETURNS

Value	Description
RC >= 0	Value of the *X•IPC* asynchronous I/O descriptor.
RC < 0	Error (see error codes below).

DESCRIPTION

XipcAsyncIoDescriptor() returns the value of the I/O descriptor being used by *X•IPC* for notifying the completion of asynchronous *X•IPC* operations initiated by the calling process. This I/O descriptor is then typically used by the process for polling on, or for multiplexing along with, other I/O descriptors.

Completion notification of an *X•IPC* asynchronous operation is indicated as a data-available event on the I/O descriptor. The process should react by running the XipcAsyncEventHandler() function. This function processes the completing asynchronous *X•IPC* operations.

The I/O descriptor may be integrated within an application's X-Window event loop environment. This is typically accomplished by passing the I/O descriptor to the XtAddInput() ot XtAppAddInput() Xt library function. The application must then be coded to call XipcAsyncEventHandler() at some point within the Xt callback function associated with the I/O event.

Refer to the Technical Note “Using I/O Descriptors for Asynchronous Operations on UNIX” in the Appendix to the *X•IPC* User Manual for further information.

ERRORS

Code	Description
XIPC_ER_NOACCESS	Process not using *X•IPC* asynchronous I/O descriptor method.

XIPC_ER_SYSERR	An internal error has occurred while processing the request.

3.4 XipcConnect() - Connect to a Login

NAME

XipcConnect() - Connect to a Login

SYNTAX

```
#include "xipc.h"


XINT
XipcConnect(InstName, Uid)

CHAR *InstName;
XINT Uid;
```

PARAMETERS

Name	Description
InstName	A pointer to a string that identifies the target instance of the XipcConnect request. *InstName* must be null terminated and must not exceed XIPC_LEN_PATHNAME characters.
Uid	The Uid of the login to connect to. The *Uid* was returned by a call to XipcLogin(). When the target login is uniquely identified by the instance name, the *Uid* argument may be specified as XIPC_USER_NULL.

RETURNS

Value	Description
RC >= 0	Connect successful.
RC < 0	Error (see error codes below).

DESCRIPTION

XipcConnect() connects the calling program to the specified *X•IPC* login identified by an *X•IPC* instance name and a *Uid*. The target login must have been created previously by the calling program via a call to XipcLogin().

Following a successful call to XipcConnect(), the specified login becomes the caller's *current login* for all subsequent *X•IPC* API calls.

The login that the user connects to must be in the user's *login working set*. The user must not be currently connected to any login.

InstName must specify the instance name exactly as it was specified to XipcLogin().

Uid is the Uid returned by XipcLogin(). If the login to connect to is uniquely identified by the instance name, meaning that the calling program logged into the target instance only once, the *Uid* argument can be specified as `XIPC_USER_NULL`.

ERRORS

Code	Description
`XIPC_ER_AMBIGUOUS`	The *Uid* specified as `XIPC_USER_NULL` but the login cannot be identified uniquely.
`XIPC_ER_BADCONFIGNAME`	The *InstName* specification is invalid or missing.
`XIPC_ER_BUSY`	User already connected to an *X•IPC* instance.
`XIPC_ER_NOTFOUND`	Entry not found in the *login working set*.
`XIPC_ER_SYSERR`	An internal error has occurred while processing the request.

INTERACTIVE COMMAND

SYNTAX

`xipcconnect` *`[InstanceName] [UserId]`*

ARGUMENTS

`InstanceName`	Name of instance to connect to: Either an instance configuration file name or an instance name (local or network) starting with the character '@'. Instance name can be specified as '*' in which case the value of the environment variable XIPC will be used. The instance name must be specified exactly as it was specified in the xipclogin command.
`UserId`	User id as returned by xipclogin

EXAMPLES

```
xipc> # Log into stand-alone instance.
      # Disconnect from the login.
      # Then reconnect to the login.

xipc> xipclogin /usr/xipc/test Joe
      Uid = 11

xipc> xipcdisconnect
      RetCode = 0

xipc> xipcconnect /usr/xipc/test 11
      RetCode=0
```

3.5 XipcDisconnect() - Disconnect From the Current Login

NAME

`XipcDisconnect()` - Disconnect from the Current Login

SYNTAX

```
#include "xipc.h"


XINT
XipcDisconnect()
```

PARAMETERS

None

RETURNS

Value	Description
RC >= 0	Disconnect successful.
RC < 0	Error (see error codes below).

DESCRIPTION

XipcDisconnect() disconnects the calling user from the user's *current login*. Unlike XipcLogout(), the XipcDisconnect() function does not terminate the user's login status. All resources held by the user entry within the instance continue to be held. All pending asynchronous operations initiated by the user entry remain pending

Following the XipcDisconnect() call, the calling program's *current login* is undefined. All *X•IPC* APIs requiring a valid *current login* will fail until the user issues an XipcConnect() or XipcLogin() call.

ERRORS

Code	Description
XIPC_ER_NOTCONNECTED	User not connected to an instance.
XIPC_ER_SYSERR	An internal error has occurred while processing the request.

INTERACTIVE COMMAND

SYNTAX

xipcdisconnect

ARGUMENTS

None

EXAMPLES

```
xipc> xipclogin /usr/xipc/test Joe
      Uid = 11

xipc> xipcdisconnect
      RetCode = 0

xipc> xipclogin /usr/xipc/test2 Joe
      Uid = 7
```

3.6 XipcError() - X•IPC Error Code Translation Function

NAME

XipcError() - X•IPC Error Code Translation Function

SYNTAX

```
#include "xipc.h"


CHAR *
XipcError(ErrCode)

XINT ErrCode;
```

PARAMETERS

Name	Description
ErrCode	The X•IPC error code whose translation is desired.

RETURNS

Value	Description

A character string pointer (see description below).

DESCRIPTION

XipcError() returns a pointer to a static character string containing a brief translation of the error code it is passed. It returns a pointer to an appropriate message for undefined error codes.

INTERACTIVE COMMAND

SYNTAX

xipcerror *ErrorCode*

ARGUMENTS

ErrorCode *X•IPC* error code

EXAMPLES

```
xipc> xipcerror -1003
      Configuration capacity limit exceeded
```

3.7 XipcFreeze() - Freeze An Instance

NAME

XipcFreeze() - Freeze An Instance

SYNTAX

```
#include "xipc.h"


XINT
XipcFreeze()
```

PARAMETERS

None.

RETURNS

Value	Description
RC >= 0	XipcFreeze successful.
RC < 0	Error (see error codes below).

DESCRIPTION

XipcFreeze() freezes the *X•IPC* instance. This function provides the same functionality as calling QueFreeze(), SemFreeze() and MemFreeze() as a unit.

For a detailed description of this function, see the description of the subsystem freeze functions.

ERRORS

See errors for the freeze functions in the subsystem documentation.

INTERACTIVE COMMAND

SYNTAX

xipcfreeze

ARGUMENTS

None.

EXAMPLES

```
xipc> xipcfreeze
      RetCode = 0
```

3.8 XipcGetOpt() - Get Parameters

NAME

`XipcGetOpt()` - To obtain the value of the various parameters

SYNTAX

```
#include "xipc.h"


XINT
XipcGetOpt(...)

... Option;
```

PARAMETERS

Name	Description

`Option` This is a macro that describes the option being set. The following table describes the options that are supported:

Option	Arguments	Description	Default Value
`XIPC_GETOPT_CONNECTTIMEOUT`	Timeout value (in seconds)	Set network connect timeout. *This value must be set **before** login.*	0
`XIPC_GETOPT_RECVTIMEOUT`	Timeout value (in milliseconds)	Set network receive timeout. NOTE: Any number less than 60 will be read as *seconds* (in order to support backward compatibility).	None
`XIPC_GETOPT_PINGTIMEOUT`	Timeout value (in milliseconds)	Set network ping timeout	--
`XIPC_GETOPT_PINGRETRIES`	Number of retries	Set network ping retry number	--
`XIPC_GETOPT_PINGFUNCTION`	Function pointer	Set the function to be invoked by *X•IPC* to determine whether a node is alive.	None
`XIPC_GETOPT_PRIVATEQUEUE`	None	Create a private IPC queue for async activity for each thread logging into *X•IPC* (for UNIX only).	Common IPC queue
`XIPC_GETOPT_MAXTEXTSIZE`	Network TextSize	Overrides the `MAX_TEXTSIZE` parameter specified in the configuration file.	Instance `MAX_TEXTSIZE`
`XIPC_GETOPT_ASYNCFD`	None	Overrides the use of signals for asynchronous operations on UNIX. *This value must be set **before** login.*	--

RETURNS

Value	Description
RC > 0	Operation succeeded.
RC < 0	Error (see error codes below).

DESCRIPTION

The function XipcGetOpt() may be invoked at the same places that XipcSetOpt() is invoked.

Example:

```
XINT PingRetries;
RetCode = XipcGetOpt(XIPC_GETOPT_PINGRETRIES(&PingRetries));

XINT TimeOut;
RetCode = XipcGetOpt(XIPC_GETOPT_RECVTIMEOUT(&Timeout));
```

The following table summarizes how the XipcGetOpt() function behaves in each situation:

Invocation Timing	
When *not* connected to an *X•IPC* default login	When connected to an *X•IPC* default login
Gets *X•IPC* default settings	Gets current login settings

Refer to the section on the Network Timeout Detection Mechanism in the Advanced Topics chapter of the *X•IPC* User Guide for details on using *X•IPC*'s Network Timeout Detection Mechanism.

ERRORS

Code	Description
XIPC_ER_OPTIONNOTSET	Option specified has not been set.
XIPC_ER_INVALIDSETOPTION	Invalid Option specified to function.
XIPC_ER_SYSERR	An internal error has occurred while executing the operation.

INTERACTIVE COMMAND

SYNTAX

xipcgetopt [*Option*]

ARGUMENTS

[*Option*]	One from the following options: CONNECTTIMEOUT, RECVTIMEOUT, PINGTIMEOUT, PINGRETRIES, PINGFUNCTION, PRIVATEQUEUE, MAXTEXTSIZE, ASYNCFD

EXAMPLES

```
xipc> xipcgetopt pingtimeout
      Parameter [pingtimeout] –> : [5]
```

3.9 XipcIdleWatch() - Control Idle Watch Monitoring

NAME

`XipcIdleWatch()` - Control Idle Watch Monitoring With Current Instance

SYNTAX

```
#include "xipc.h"


XINT
XipcIdleWatch(WatchOption)

XINT WatchOption;
```

PARAMETERS

Name	Description
`WatchOption`	This parameter can be set to one of the following values:

- ☐ The `XIPC_IDLEWATCH_START` value notifies the instance to start monitoring the calling user as part of the instance's idle user detection activity.
- ☐ The `XIPC_IDLEWATCH_STOP` value notifies the instance to stop monitoring the calling user as part of the instance's idle user detection activity.
- ☐ The `XIPC_IDLEWATCH_MARK` argument value notifies the instance that the calling user is still alive. Calling XipcIdleWatch() with this option is a means of issuing a "heartbeat" to the instance. This saves the user from detection during the current idle-period cycle. Another way of viewing this option is as if the user is executing a null *X•IPC* operation within the instance.

RETURNS

Value	Description
RC = 0	Operation succeeded.
RC < 0	Error (see error codes below).

DESCRIPTION

The XipcIdleWatch() function call can be used to toggle the user's state within an instance between being watched and not being watched. It is additionally possible for a user to notify the `xipcidld` daemon that it is still alive even though it has not recently performed X•IPC operations within the instance.

By default, users logging into an X•IPC instance that is being monitored for idle users are *not* subject to monitoring. Users wishing to have their activity within the instance monitored *must* notify the `xipcidld` daemon of such a desire.

Refer to the Technical Notes Appendix in the X•IPC User Guide for details on using X•IPC Idle User Detection Mechanism.

ERRORS

Code	Description
XIPC_ER_NOTLOGGEDIN	User not logged in to an X•IPC instance.
XIPC_ER_SYSERR	An internal error has occurred while executing the operation.
XIPCNET_ER_CONNECTLOST	Connection to instance lost.
XIPCNET_ER_SYSERR	Operating system error.

INTERACTIVE COMMAND

SYNTAX

`xipcidlewatch`

ARGUMENTS

`Option` One of “start,” “stop” or “mark.”

EXAMPLES

```
xipc> xipcidlewatch start
      RetCode = 0
```

3.10 XipcInfoLogin() - Get Login Information

NAME

`XipcInfoLogin()` - Get Login Information

SYNTAX

```
#include "xipc.h"


XINT
XipcInfoLogin(InfoLogin, NumEnt, Cursor)

XIPCINFOLOGIN *InfoLogin;
XINT NumEnt;
XINT *Cursor;
```

PARAMETERS

Name	Description
Infologin	A pointer to a structure or an array of structures of type XIPCINFOLOGIN. If the parameter is `NULL`, the current number of logins is returned.
NumEnt	Number of entries in the *InfoLogin* array.
Cursor	A pointer to a cursor variable used by *X•IPC* between successive calls during enumeration of the logins. If the cursor pointer is set to `NULL`, the *current login* information is returned.

RETURNS

Value	Description
RC > 0	Actual number of login information structures returned.
RC = 0	No login information returned.

DESCRIPTION

XipcInfoLogin() provides information about the calling process *login working set*. The information about a Login is returned in an XIPCINFOLOGIN structure, which is defined as follows:

```
typedef struct _XIPCINFOLOGIN    /* Login Information */
{
   XINT Lid;                     /* Login Id */
   CHAR FAR InstanceName;        /* Pointer to instance name */
   XINT UserId;                  /* User Id */
}
XIPCINFOLOGIN;
```

XipcInfoLogin() provides the capability to enumerate all the logins in the process' *login working set* and to get information either on one login or on a set of logins every time it is called.

A cursor variable, defined by the caller, is used by X•IPC to maintain its position within the *login working set* between successive calls. The cursor variable must be initialized to the value XIPC_LOGIN_INIT_ENUMERATION before the first call to XipcInfoLogin().

A number of macros that simplify the interface to XipcInfoLogin are provided and can be used for attaining information from XipcInfoLogin.

XipcInfoLogin allows the selection of the returned information as follows:

- The current login. This information is returned when the *NumEnt* argument is set to 1 and the *Cursor* argument is set to NULL. Alternatively, the macro XIPC_LOGIN_CURRENT can be used as follows:

  ```
  NumLogins = XipcInfoLogin(InfoLogin, XIPC_LOGIN_CURRENT)
  ```

- Number of logins. This information is returned when *InfoLogin* is set to NULL. Alternatively, the macro XIPC_LOGIN_COUNT can be used as follows:

  ```
  NumLogins = XipcInfoLogin(XIPC_LOGIN_COUNT)
  ```

- Enumeration of the logins in the process' *login working set*. Each call returns the information of the next login in the set. This information is returned when the *NumEnt* argument is set to 1 and the *Cursor* argument points to a cursor variable. Alternatively, the macro XIPC_LOGIN_NEXT can be used as follows:

  ```
  c = XIPC_LOGIN_INIT_ENUMERATION;
  while (XipcInfoLogin(InfoLogin, XIPC_LOGIN_NEXT(c)) > 0)
  {
    /* Process InfoLogin */
  }
  ```

- A list of logins. This information is returned when *InfoLogin* points to an array of login information structures, *NumEnt* is the number of entries in the structure and *Cursor* points to a cursor variable.

ERRORS

None

INTERACTIVE COMMAND

SYNTAX

xipcinfologin

ARGUMENTS

None

EXAMPLES

```
xipc> xipcinfologin
      Uid    Instance
      ---    --------
      11    /usr/xipc/test
       7    @Server
      31    @DBServer
```

3.11 XipcInfoSystemError() - Get Additional System Error Information

NAME

`XipcInfoSystemError()` - Get Additional System Error Information

SYNTAX

```
#include "xipc.h"


XINT
XipcInfoSystemError(XIPC_SYSERR *XipcSysErr);
```

PARAMETERS

Name	Description
`XipcSysErr`	A pointer to a structure of type `XIPC_SYSERR`, into which System Error information will be copied.

RETURNS

Value	Description
RC >= 0	Successful.
RC < 0	Error (see error codes below).

DESCRIPTION

XipcInfoSystemError() takes as an argument a pointer to a structure of type `XIPC_SYSERR` into which the information regarding the last occurrence of a system error is copied. The structure is defined as follows:

```
typedef struct XIPC_SYSERR
{
    XINT Status1;                   /* First status code */
    XINT Status2;                   /* Second status code in case of some O.S.*/
    XINT Source;                    /* Operating System or Network error*/
    XINT Class;                     /* XIPC's best guess on type of error
                                        XIPC_ER_OTHER or XIPC_ER_CONNECTION /
    CHAR OperatingSystem[XIPC_LEN_FUNCNAME]; /* Name of operating system */
    XINC Location;                          /* Local or remote Error /
    CHAR XipcVerb[XIPC_LEN_FUNCNAME];       /* XIPC verb reporting the error */
    CHAR SystemCall[XIPC_LEN_FUNCNAME];     /* Name of failing system service */
    CHAR ErrorMessage[XIPC_LEN_ERRORMSG];   /* Formatted error message */
}
XIPC_SYSERR;
```


ERRORS

Code	Description
XIPC_ER_BADBUFFER	*MsgBuf* is NULL.
XIPC_ER_LOGINNOTFOUND	User not logged into instance (User never logged in, was aborted or disconnected).
XIPC_ER_MEMORY	Out of system memory.

3.12 XipcInfoUser() - Get User Information

NAME

`XipcInfoUser()` - Get User Information

SYNTAX

```
#include "xipc.h"


XINT
XipcInfoUser(Uid, InfoUser)

XINT Uid;
XIPCINFOUSER *InfoUser;
```

PARAMETERS

Name	Description
`Uid`	The user ID of the user whose information is desired. The user ID may be specified as `XIPC_INFO_NEXT(`*`Uid`*`)`, in which case information regarding the first active user whose user ID is greater or equal to the specified *Uid* is returned.
`InfoUser`	Pointer to a structure of type XIPCINFOUSER, into which the user information will be copied.

RETURNS

Value	Description
RC >= 0	Successful.
RC < 0	Error (see error codes below).

DESCRIPTION

XipcInfoUser() fills the specified structure (`*InfoUser`) with information about the user identified by *Uid*. The user may not be an asynchronous user. The structure is defined as follows:

```
typedef struct _XIPCINFOUSER
{
   XINT Uid;                              /* User Id */
   XINT Pid;                              /* Process Id of User */
   XTID Tid;                              /* Thread Id of User */
   XINT LoginTime;                        /* Time of login to instance */
   CHAR Name[XIPC_LEN_XIPCNAME + 1];      /* User login name */
   CHAR NetLoc[XIPC_LEN_NETLOC + 1];      /* Name of client node */
 }
XIPCINFOLOGIN;
```

ERRORS

Code	Description
XIPC_ER_BADUID	Invalid *Uid* parameter.
XIPC_ER_NOTLOGGEDIN	User not logged into instance (User never logged in, was aborted or disconnected).
XIPCNET_ER_CONNECTLOST	Connection to instance lost.
XIPCNET_ER_NETERR	Network transmission error.
XIPCNET_ER_SYSERR	Operating system error.

INTERACTIVE COMMAND

SYNTAX

xipcinfouser *UserId* | **first** | next(*UserId*)

ARGUMENTS

UserId Print info on the **first** user, the user with user id *Uid* or the **next** higher user id.

EXAMPLES

```
xipc> xipcinfouser
      UID:  7   Name:  "LockManager"   Pid = 23795    Tid = 0
      Host: '*LOCAL*'   Login Time:    Wed Sep 24 11:24:48   2003
```

3.13 XipcInfoVersion() - Get X•IPC Version Information

NAME

XipcInfoVersion() - Get X•IPC version information

SYNTAX

```
#include "xipc.h"

XINT
XipcInfoVersion(
      XIPCINFOVERSION *InfoVersion)
```

PARAMETERS

Name	Description
InfoVersion	Pointer to a structure of type XIPCINFOVERSION where general X•IPC version data will be returned.

RETURNS

Value	Description
RC >= 0	Operation successful.
RC < 0	Error (see error codes below).

DESCRIPTION

The XipcInfoVersion() function is used to acquire X•IPC version data. The function takes, as its one argument, a pointer to a data structure of type XIPCINFOVERSION. This data structure, when passed to the function, returns populated with version information about the currently running X•IPC environment.

The XIPCINFOVERSION data structure is defined as follows:


```
typedef struct _XIPCINFOVERSION
{
    XINT    Major;                              /* Such as 3 from 3.1.0 */
    XINT    Minor;                              /* Such as 1 from 3.1.0 */
    XINT    Level;                              /* Such as 0 from 3.1.0 */
    CHAR    Modifier[XIPC_LEN_XIPCNAME +1];     /* Optional string such as "(GA)" */
                                                /* May be NULL. (See example below) */
    CHAR    BuildCommon;                        /* Such as 'b' in 3.3.0.ba */
    CHAR    BuildSpecific;                      /* Such as 'a' in 3.3.0.ba */
    CHAR    OperatingSystem[XIPC_LEN_XIPCNAME +1];/* Such as "Windows NT 4.0" */
    CHAR    StringSummary[XIPC_LEN_XIPCNAME +1]; /* Summary of above data */

} XIPCINFOVERSION;
```

ERRORS

Code	Description
MOM_ER_BADBUFFER	Return buffer pointer is NULL.

INTERACTIVE COMMAND

SYNTAX

```
xipcinfoversion | xipcver
```

ARGUMENTS

None

EXAMPLES

```
xipc>  xipcinfoversion
       XIPC Version 3.4.0 aa - Windows NT 4.0
```

3.14 XipcInit() – Initiate the X•IPC Platform Environment

NAME

XipcInit() – Initiate the X•IPC Platform Environment

SYNTAX

```
#include "xipc.h"


XINT
XipcInit()
```

PARAMETERS

None

RETURNS

Value	Description
RC >= 0	Platform initiated successfully.
RC < 0	Error (see error codes below).

DESCRIPTION

XipcInit() is used for initializing the X•IPC environment on a computer platform from under program control. Alternatively, the `xipcinit` command can be executed from the operating system prompt. XipcInit() sets up all internal structures and background processes needed for supporting X•IPC activity on the platform.

XipcInit() references the `xipc.env` file located within the `XIPCROOT` location on the platform's file system when it is invoked. The `xipc.env` file provides the list of *program* and *catalog* parameters that are to be set up by XipcInit(). The overall syntax for `xipc.env,` within a TCP/IP setting, as well as *program* and *catalog* parameter definitions can be found in the description of the `xipcinit` command.

ERRORS

Code	Description
MEM_ER_FAILSTART	MemSys initialization failed.
MEM_ER_GHOSTSTART	Cannot register MemSys with X•IPC object daemon.
QUE_ER_FAILSTART	QueSys initialization failed.
QUE_ER_GHOSTSTART	Cannot register QueSys with X•IPC object daemon.

XIPC_ER_BUSY	User already connected to an *X•IPC* instance.
XIPC_ER_FILEOPEN	Error opening file.
XIPC_ER_GHOSTSTART	Cannot register with *X•IPC* object daemon.
XIPC_ER_INSTACTIVE	Instance is already active.
XIPC_ER_NOTLOCAL	XipcStart() was called as part of an *X•IPC*/Network program.
XIPC_ER_NOPLATFORMFILE	Platform configuration file cannot be opened.
XIPC_ER_NOROOTDIR	Cannot locate *X•IPC* root directory.
XIPC_ER_SYSERR	An internal error has occurred while processing the request.

INTERACTIVE COMMAND

SYNTAX

xipcinit

ARGUMENTS

None

EXAMPLES

```
xipc> xipcinit
      xipcinit:XIPC Platform Environment Initiated
      Win32 – XIPC 3.4.0aa  [Build 5012]
      RetCode = 0
```

3.15 XipcListXXX() - List Active Network Instances

NAME

XipcListStart() - Start an Instance Search

XipcListNext() - Get Information for Next Instance

XipcListEnd() - End Instance Search

SYNTAX

```
#include "xipc.h"


XINT
XipcListStart(HandlePtr, HostName)
XANY **HandlePtr;
CHAR *HostName

XINT
XipcListNext(Handle, InstanceInfo)
XANY *Handle;
XIPCINFOINSTANCE *InstanceInfo;

XINT
XipcListEnd(Handle)
XANY *Handle;
```

PARAMETERS

Name	Description
Handle	A pointer to an internal X•IPC instance list data structure. The pointer is initialized passing its address (*HandlePtr* is &Handle) to a call of XipcListStart(); it must be provided on all subsequent calls to XipcListNext() and XipcListEnd().
HandlePtr	A pointer to *Handle* (see above).
HostName	A character string that contains the name of the network node whose instances are to be listed. If the argument is specified as NULL, the XIPCHOST, XIPCHOSTLIST, XIPCCAT and XIPCCATLIST environment variables are used to determine the instance search range.
InstanceInfo	A pointer to a structure that is populated with instance information as a result of every successful XipcListNext() call.

RETURNS

Value	Description
RC > 0	Operation succeeded.
RC = 0	End of instance list (XipcListNext() only).
RC < 0	Error (see error codes below).

DESCRIPTION

The XipcListXxx() set of function calls are used to obtain a list of registered instances on the network. XipcListStart() is called to initialize an instance search. After a successful XipcListStart() call, a sequence of calls to XipcListNext() provide information about registered instances, one instance per call. An XipcListEnd() call terminates the instance search and releases resources.

The XipcListXxx() function calls all use a user defined handle which is a pointer to an internal *X•IPC* data structure that is used by *X•IPC* to control the instance search. The XipcListStart() function call initializes the handle. Subsequent calls to XipcListNext() and XipcListEnd() use the same handle.

The list functions provide one of two kinds of lists:

- A single host list provides a list of all instances started on a single node. This list is generated when the *HostName* argument specifies the name of a node (i.e., it is not `NULL`).
- A complete list provides a list of all instances within the calling process's instance search range. (Refer to section 2.2.2 for a discussion of search criteria and priorities.)

The information provided for each instance is placed in the following structure:

```
typedef struct _XIPCINFOINSTANCE
{
    XINT BufSize;                       /* Instance network buffer size */
    CHAR ConfigName[XIPC_LEN_PATHNAME]; /* Instance config file name */
    CHAR Name[XIPC_LEN_NETNAME];        /* Instance network name */
    CHAR HostName[XIPC_LEN_NETNAME];    /* Instance host name */
}
XIPCINFOINSTANCE;
```

ERRORS

Code	Description
XIPC_ER_MEMORY	Out of system memory
XIPC_ER_PLATNOTACTIVE	Platform instance not active.
XIPC_ER_SYSERR	An internal error has occurred while executing the operation
XIPCNET_ER_NOSEARCH	Network instance search range undefined
XIPCNET_ER_SESSION	Cannot start session with *X•IPC* server
XIPCNET_ER_NETERR	Network transmission error

INTERACTIVE COMMAND

SYNTAX

xipclist *[NodeName]*

ARGUMENTS

NodeName Name of node about which xipclist's reporting should be limited.

EXAMPLES

```
xipc> xipclist
      Machine.................[grumpy]
      Instance Name...........[server]
      Instance File Name......[/xipc/server]
      Maximum Text Size.......[1024]
```

3.16 XipcLogin() - Log Into an Instance

NAME

`XipcLogin()` - Log Into an Instance

SYNTAX

```
#include "xipc.h"


XINT
XipcLogin(InstName, UserName)

CHAR *InstName;
CHAR *UserName;
```

PARAMETERS

Name	Description
`InstName`	A pointer to a string that identifies the target instance of the XipcLogin request. *InstName* must be null terminated and must not exceed `XIPC_LEN_PATHNAME` characters.
`UserName`	A pointer to a string that contains a user Login name. *UserName* must be null terminated, and must not exceed `XIPC_LEN_USERNAME` characters. Duplicate *UserName*s are permitted. If the *UserName* is specified as `XIPC_LOGIN_SUPERUSER`, then the user is logged in as a Super User.

RETURNS

Value	Description
RC >= 0	Login successful. RC is the Uid of the user.
RC < 0	Error (see error codes below).

DESCRIPTION

XipcLogin() logs the calling user, identified as *UserName*, into the instance identified by *InstName*. It also sets that login as the user's *current login*.

InstName specifies the target instance in one of three forms:

- A stand-alone instance is specified using its instance file name.

- A local instance is specified using the following format:

  @[*LocalName*]

  where:

  LocalName is the name assigned to the instance when it was started.

- A network instance is specified using the following format:

  @[*NodeName*:]*NetworkName*

  where:

  NodeName is the name of the node where the instance resides. The *NodeName* part is optional and when specified, it must be followed by a colon.

  NetworkName is the instance network name assigned to the instance when it was started.

The requested instance must have been started previously. The user must not be currently connected to an instance. XipcLogin() must succeed before any other X•IPC functions can be invoked.

If the *UserName* is specified as XIPC_LOGIN_SUPERUSER, the user is logged in as Super User. A Super User login can be used to access a "corrupt" instance or an instance that has exhausted some of its resources, preventing regular users from logging in.

ERRORS

Code	Description
MEM_ER_CAPACITY_NODE	MemSys node table full.
MEM_ER_CAPACITY_USER	MemSys user table full.
MOM_ER_CAPACITY_ASYNC_USER	MomSys async user table full.
MOM_ER_CAPACITY_USER	MomSys user table full.
QUE_ER_CAPACITY_NODE	QueSys node table full.
QUE_ER_CAPACITY_USER	QueSys user table full.
SEM_ER_CAPACITY_NODE	SemSys node table full.
SEM_ER_CAPACITY_USER	SemSys user table full.
XIPC_ER_BADCONFIGFILE	The *InstName* file is inaccessible.
XIPC_ER_BADCONFIGINFO	The contents of the *InstName* file is erroneous.
XIPC_ER_BADCONFIGNAME	The *InstName* specification is invalid or missing.
XIPC_ER_BADUSERNAME	Invalid *UserName* parameter.
XIPC_ER_CAPACITY_THREAD	Thread table full.
XIPC_ER_BUSY	User already connected to an X•IPC instance.
XIPC_ER_NOACCESS	Unable to connect to X•IPC.
XIPC_ER_SYSERR	An internal error has occurred while processing the request.
XIPCNET_ER_BADNETNAME	Network name missing "@" prefix.

XIPCNET_ER_NOFILE	Cannot open “XIPCHOSTLIST” or “XIPCCATLIST” file.
XIPCNET_ER_NOFORK	Process limit exceeded on server.
XIPCNET_ER_NOSEARCH	Instance search range not defined.
XIPCNET_ER_NOTFOUND	*InstName* instance not found within instance search range.
XIPCNET_ER_SESSION	Cannot connect to *InstName* instance.

INTERACTIVE COMMAND

SYNTAX

xipclogin *[InstanceName] [UserName]*

ARGUMENTS

InstanceName — Name of instance to log into: Either an instance file name or an instance name (local or network) starting with the character '@'. Instance name can be specified as '*' in which case the value of the environment variable XIPC will be used.

UserName — Name to be assigned to the *X•IPC* user. If omitted, the string XIPC will be used. If the name SUPERUSER is used, the user is logged in as a SUPERUSER.

EXAMPLES

```
xipc> xipclogin  /tmp/config George
      Uid = 11

xipc> # Log into instance “Server”. “xipc” is default user name
xipc> xipclogin  @Server
      Uid = 1

xipc> # Log into network instance on node “dopey”
xipc> xipclogin  @dopey:Server George
      Uid = 1
```

3.17 XipcLogout() - Log Out of an Instance

NAME

`XipcLogout()` - Log Out of an Instance

SYNTAX

```
#include "xipc.h"


XINT
XipcLogout()
```

PARAMETERS

None.

RETURNS

Value	Description
RC >= 0	Logout successful.
RC < 0	Error (see error codes below).

DESCRIPTION

XipcLogout() logs the calling user out of the instance it is currently logged into. Resources held by the user are released as follows:

- All *X•IPC* semaphores held by the user are released.
- All *X•IPC* memory sections held by the user are released.

Following the XipcLogout() call, the calling program's *current login* is undefined. All *X•IPC* APIs requiring a valid *current login* will fail until the user issues an XipcConnect() or XipcLogin() call.

ERRORS

Code	Description
`XIPC_ER_NOTLOGGEDIN`	User not logged into *X•IPC* (User never logged in, is disconnected or was aborted).
`XIPC_ER_SYSERR`	An internal error has occurred while processing the request.

XIPCNET_ER_CONNECTLOST	Connection to instance lost.
XIPCNET_ER_NETERR	Network transmission error.
XIPCNET_ER_SYSERR	Operating system error.

INTERACTIVE COMMAND

SYNTAX

xipclogout

ARGUMENTS

None.

EXAMPLES

```
xipc> xipclogout
      RetCode = 0
```

3.18 XipcMaskTraps() - Activate Trap Mask

NAME

XipcMaskTraps() - Activate Trap Mask

SYNTAX

```
#include "xipc.h"

XINT
XipcMaskTraps()
```

PARAMETERS

None.

RETURNS

Value	Description
RC >= 0	XipcMaskTraps successful.
RC < 0	Error (see error codes below).

DESCRIPTION

XipcMaskTraps() activates the calling user's trap function mask. This mask, when active, is detected by the XIPC_TRAP_FUNCTION_TEST() macro.

The XIPC_TRAP_FUNCTION_TEST() macro is typically placed at the start of a process' trap service functions, and serves as a guard against trap service function execution at times when the subject process must not be disturbed (refer to the XIPC_TRAP_FUNCTION_TEST() manual pages for details). *X•IPC* uses the XipcMaskTraps() mechanism internally to temporarily prevent trap function execution during internal *X•IPC* processing. Note that the notification of asynchronous operation completion is blocked, as well, while the mask is active.

Trap function masking remains in effect until a bracketing XipcUnmaskTraps() call is issued–at which point the mask is removed and pending trap functions and asynchronous operation notifications are processed.

ERRORS

Code	Description
XIPC_ER_NOTLOGGEDIN	User not logged into *X•IPC* (User never logged in, is disconnected or was aborted).
XIPC_ER_ISMASKED	Trap mask already active.

INTERACTIVE COMMAND

SYNTAX

xipcmasktraps

ARGUMENTS

None.

EXAMPLES

```
xipc> xipcmasktraps
      RetCode = 0
```

3.19 XipcPing() - Detect Remote Host

NAME

`XipcPing()` - To detect whether a remote host is reachable or not

SYNTAX

```
#include "xipc.h"


XINT
XipcPing(HostName, PingTimeOut, PingRetries)

CHAR HostName;
XINT PingTimeOut;
XINT PingRetries;
```

PARAMETERS

Name	Description
HostName	The name of the remote host.
PingTimeOut	Ping Timeout value in milliseconds.
PingRetries	The number of times to retry pinging the host in case of failure.

RETURNS

Value	Description
RC > 0	Operation succeeded.
RC < 0	Error (see error codes below).

DESCRIPTION

The XipcPing() function call can be invoked by any *X•IPC* application to detect whether a remote host is reachable or not. For this API to function, **the `xipciad` daemon must be running on the remote host**.

Refer to the section on the Network Timeout Detection Mechanism in the Advanced Topics chapter of the *X•IPC* User Guide for details on using *X•IPC*'s Network Timeout Detection Mechanism.

ERRORS

Code	Description
XIPC_ER_SYSERR	An internal error has occurred while executing the operation.
XIPCNET_ER_BADHOST	Host Name provided is not valid.
XIPCNET_ER_HOSTUNREACHABLE	Host may be unreachable.

3.20 XipcSetOpt() - Change Parameter Defaults

NAME

`XipcSetOpt()` - To change the default values of the various parameters

SYNTAX

```
#include "xipc.h"


XINT
XipcSetOpt(Option)

... Option;
```

PARAMETERS

Name **Description**

`Option` This is a macro that describes the option being set. The following table describes the options that are supported:

Option	Arguments	Description	Default Value
`XIPC_SETOPT_CONNECTTIMEOUT`	Timeout value (in seconds)	Set network connect timeout. *This value must be set **before** login.*	0
`XIPC_SETOPT_RECVTIMEOUT`	Timeout value (in milliseconds)	Set network receive timeout. NOTE: Any number less than 60 will be read as *seconds* (in order to support backward compatibility).	None
`XIPC_SETOPT_PINGTIMEOUT`	Timeout value (in milliseconds)	Set network ping timeout	--
`XIPC_SETOPT_PINGRETRIES`	Number of retries	Set network ping retry number	--
`XIPC_SETOPT_PINGFUNCTION`	Function pointer	Set the function to be invoked by *X•IPC* to determine whether a node is alive.	None
`XIPC_SETOPT_PRIVATEQUEUE`	None	Create a private IPC queue for async activity for each thread logging into *X•IPC* (for UNIX only).	Common IPC queue
`XIPC_SETOPT_MAXTEXTSIZE`	Network TextSize	Overrides the `MAX_TEXTSIZE` parameter specified in the configuration file.	Instance `MAX_TEXTSIZE`
`XIPC_SETOPT_ASYNCFD`	None	Overrides the use of signals for asynchronous operations on UNIX. *This value must be set **before** login.*	--

Examples:

```
RetCode = XipcSetOpt(XIPC_SETOPT_PINGTIMEOUT(2000));

RetCode = XipcSetOpt(XIPC_SETOPT_PINGRETRIES(1));

RetCode = XipcSetOpt(XIPC_SETOPT_RECVTIMEOUT(5));

RetCode = XipcSetOpt(XIPC_SETOPT_PINGFUNCTION(MyPingFunc));
```

where:

`MyPingFunc` is a user-defined Ping Function. This function should return `1` on success and `-1` on failure.

The format for this user-defined function is:

```
XINT MyPingFunc (CHAR *IP_Address)
{
    XINT PingResult;

    /***** Ping Implementation ******/

    return(PingResult);
}

RetCode = XipcSetOpt (XIPC_SETOPT_MAXTEXTSIZE(100000));

RetCode = XipcSetOpt (XIPC_SETOPT_MAXTEXTSIZE(0)):
```

Setting MAXTEXTSIZE=0 resets the Network TextSize to use the default MAX_TEXTSIZE parameter from the `XIPC` configuration file.

RETURNS

Value	Description
RC > 0	Operation succeeded.
RC < 0	Error (see error codes below).

DESCRIPTION

The XipcSetOpt() function call can be invoked to change the default values of the various parameters. If the parameters are modified **before** login, then it will affect all the corresponding logins for that specific thread. If the parameters are modified **after** login then only that particular login will be affected by this change.

Refer to the section on the Network Timeout Detection Mechanism in the Advanced Topics chapter of the *X•IPC* User Guide for details on using *X•IPC*'s Network Timeout Detection Mechanism.

ERRORS

Code	Description
`XIPC_ER_BUSY`	User already connected to an instance.

XIPC_ER_INVALIDSETOPTION	Invalid *Option* specified to function.
XIPC_ER_SYSERR	An internal error has occurred while executing the operation.

INTERACTIVE COMMAND

SYNTAX

xipcsetopt [*Option*]

ARGUMENTS

[*Option*]	One from the following options: CONNECTTIMEOUT, RECVTIMEOUT, PINGTIMEOUT, PINGRETRIES, PINGFUNCTION, PRIVATEQUEUE, MAXTEXTSIZE, ASYNCFD

EXAMPLES

```
xipc> xipcsetopt pingtimeout 5
      Parameter [pingtimeout] –> New value [5]
```

3.21 XipcStart() - Start an Instance

NAME

XipcStart() - Start an Instance

SYNTAX

```
#include "xipc.h"


XINT
XipcStart(InstFileName, InstName, Options)

CHAR *InstFileName;
CHAR *InstName;
... Options;
```

PARAMETERS

Name	Description
InstFileName	A pointer to a string that identifies the path of the instance configuration file. The path name must be a valid file specification in the platform where the XipcStart() function is executed.
InstName	A pointer to a string that contains a name to be assigned to an instance, or NULL if not specified. If NULL, the name will be taken from the LOCALNAME or NETNAME parameter specified in the [XIPC] section of the Instance Configuration File. (See Section 2.2.4.) Otherwise, the instance will be started as a "stand-alone" instance having no registered name. Such an instance is only accessible via its Instance File Name. (Refer to XipcLogin() for further information.)
Options	XIPC_START_LOCAL, XIPC_START_NETWORK, XIPC_START_INITIALIZE, XIPC_START_TEST, XIPC_START_REPORT or XIPC_START_NOOPT. These options can be ORed together.

RETURNS

Value	Description
RC >= 0	Instance started successfully.
RC < 0	Error (see error codes below).

DESCRIPTION

XipcStart() is used to start and initialize an instance of *X•IPC*. It must be executed before any program can log into that instance.

The specified *InstFileName* is the name of the ".cfg" file containing configuration information for this instance. Multiple instances can be started on a single platform, provided that each one uses a unique *InstFileName*. If the instance includes the MomSys subsystem, their instance file names should be in separate directories.

Note that *X•IPC* instances that are started with an assigned name (either a *Local* or a *Network* name) are visible to the `xipclist` utility command. It is sometimes desirable that an instance's existence not be visible to `xipclist`. This can be accomplished by assigning the instance a name starting with the '_' (underscore) character. So for example: an instance named `foo` would be visible to `xipclist`, while an instance named `_foo` would not.

When starting a local instance, if a local instance having the specified local name already exists on the current node, then XipcStart() fails.

When starting a network instance, if a network instance having the specified network name already exists on the network (within the user's instance search range) then XipcStart() fails. The instance search range is a function of the `XIPCHOST, XIPCHOSTLIST, XIPCCAT` and `XIPCCATLIST` environment variables. Details of *X•IPC* /Network environment variables are provided earlier in this volume. The expected format and contents of a configuration file is also detailed.

The following table summarizes the available options and their functions:

Option	Description
`XIPC_START_LOCAL`	Directs *X•IPC* to start an instance as a local instance, with `InstName` as its local name.
`XIPC_START_NETWORK`	Directs *X•IPC* to start an instance as a network instance, with `InstName` as its network name.
`XIPC_START_INITIALIZE`	Directs *X•IPC* to reinitialize the message repository. NOTE: **All data (local app-queues, remote app-queues and messages) will be destroyed.** Use this option *only* if there is a need to restart an instance with a fresh message repository. This is only applicable to the MomSys subsystem.
`XIPC_START_TEST`	Directs *X•IPC* to print a report on the standard output file that lists configuration parameters and memory requirements; the instance is *not* started. This option is useful in order to get information about the instance memory requirements without actually having the resources available.
`XIPC_START_REPORT`	Directs *X•IPC* to print a report on the standard output file that lists configuration parameters and memory requirements; the instance *is* started. The report has the same information and format as the report printed by the `xipcstart` command.
`XIPC_START_NOOPT`	This option must be used when no other option is selected.

The XipcStart() function call can only be run by an *X•IPC* /Stand-Alone program, on the platform where the instance is to be started. Otherwise it returns `XIPC_ER_NOTLOCAL`.

ERRORS

Code	Description
`MEM_ER_FAILSTART`	MemSys initialization failed.

MEM_ER_GHOSTSTART	Cannot register MemSys with *X•IPC* object daemon.
MEM_ER_NOSECCFG	No [MEMSYS] section in ".cfg" file.
MEM_ER_NOSECIDS	No [MEMSYS] section in ".ids" file.
QUE_ER_FAILSTART	QueSys initialization failed.
QUE_ER_GHOSTSTART	Cannot register QueSys with *X•IPC* object daemon.
QUE_ER_NOSECCFG	No [QUESYS] section in ".cfg" file.
QUE_ER_NOSECIDS	No [QUESYS] section in ".ids" file.
SEM_ER_FAILSTART	SemSys initialization failed.
SEM_ER_GHOSTSTART	Cannot register SemSys with *X•IPC* object daemon.
SEM_ER_NOSECCFG	No [SEMSYS] section in ".cfg" file.
SEM_ER_NOSECIDS	No [SEMSYS] section in ".ids" file.

XIPC_ER_BADCONFIGFILE	The *InstFileName* ".cfg" file is inaccessible.
XIPC_ER_BADCONFIGINFO	The contents of the *InstFileName* ".cfg" file is erroneous.
XIPC_ER_BADCONFIGNAME	The *InstFileName* specification is invalid or missing.
XIPC_ER_BADIDENTFILE	The instance ".ids" file is inaccessible.
XIPC_ER_BADTEMPFILE	The instance ".tmp" file is inaccessible.
XIPC_ER_GHOSTSTART	Cannot register with *X•IPC* object daemon.
XIPC_ER_IDLD	Cannot communicate with `xipcidld` daemon
XIPC_ER_INSTACTIVE	Instance is already active.
XIPC_ER_NOTLOCAL	XipcStart() was called as part of an *X•IPC*/Network program.
XIPC_ER_SYSERR	An internal error has occurred while processing the request.

INTERACTIVE COMMAND

SYNTAX

xipcstart *InstFileName InstName [Options]*

ARGUMENTS

InstFileName — The instance configuration file name of instance to be started (i.e., the path name of its instance configuration file). The Instance File Name can be omitted, in which case the value of the environment variable `XIPC` will be used.

InstName — Name to be assigned to the instance. The Instance Name can be omitted, in which case the optional value in the [`XIPC`] section of the Instance Configuration File may be used. (See *`[Options]`* below for setting an instance as `local` or `network`.) .) Otherwise, the instance will be started as a "stand-alone" instance having no registered name. Such an instance is only accessible via its Instance File Name. (Refer to `XipcLogin()` for further information.)

`[Options]` — One or more of the following: `initialize, network, local, report, test` or `0.` When listing multiple options, they should be separated by commas.

Note: Asterisks (*) can be used as "place holders," with the defaults noted above, if the arguments preceding *`[Options]`* are not specified. See the example below.

EXAMPLES

```
xipc> # Start a Network instance
xipc> xipcstart  /tmp/config  Server
      XipcStart(SemSys):
      .
      .
      XipcReg: Network Instance [Server] Registered.
      RetCode = 0

xipc> # Start a Stand-Alone Instance.
xipc> # Use XIPC environment variable to specify instance name.
xipc> # Do not output report.
xipc> xipcstart  * *
      RetCode = 0
```

3.22 XipcStop() - Stop an Instance

NAME

XipcStop() - Stop an Instance

SYNTAX

```
#include "xipc.h"


XINT
XipcStop(InstName, Options)

CHAR *InstName;
... Options;
```

PARAMETERS

Name	Description
InstName	A pointer to a string that identifies the instance. It may be specified as either an Instance File Name as started or as @InstName, where InstName is the name specified when the instance was started.
Options	XIPC_STOP_REPORT, XIPC_STOP_FORCE or XIPC_STOP_NOOPT.

RETURNS

Value	Description
RC >= 0	Instance stopped successfully.
RC < 0	Error (see error codes below).

DESCRIPTION

The XipcStop() function call is used to stop an instance of *X•IPC*. All operating system resources held by that instance are released, *except those message repository resources used by the MomSys subsystem.* Users of the instance which have not logged out are forcibly aborted from all subsystems.

The specified *InstName* identifies the instance to be brought down.

When the *Options* parameter is set to XIPC_STOP_REPORT, a report that lists stopped subsystems is printed on the standard output file. The report has the same information and format as the report printed by the xipcstop command. When the *Options* parameter is set to XIPC_STOP_FORCE, system resources held by the specified instance are forcefully cleaned. WARNING: This option should *only* be employed when a standard xipcstop command "crashes."

The XipcStop() function call can only be run by an *X•IPC* /Stand-Alone program, on the platform where the instance is to be started. Otherwise it returns `XIPC_ER_NOTLOCAL`.

ERRORS

Code	Description
`MEM_ER_FAILSTOP`	MemSys termination failed.
`MEM_ER_GHOSTSTOP`	Cannot deregister MemSys with *X•IPC* object daemon.
`MEM_ER_NOSECIDS`	No `[MEMSYS]` section in ".ids" file.
`QUE_ER_FAILSTOP`	QueSys termination failed.
`QUE_ER_GHOSTSTOP`	Cannot deregister QueSyswith *X•IPC* object daemon.
`QUE_ER_NOSECIDS`	No `[QUESYS]` section in ".ids" file.
`SEM_ER_FAILSTOP`	SemSys termination failed.
`SEM_ER_GHOSTSTOP`	Cannot deregister SemSys with *X•IPC* object daemon.
`SEM_ER_NOSECIDS`	No `[SEMSYS]` section in ".ids" file.
`XIPC_ER_BADCONFIGFILE`	The *InstFileName* file is inaccessible.
`XIPC_ER_BADCONFIGINFO`	The contents of the *InstFileName* file is erroneous.
`XIPC_ER_BADCONFIGNAME`	The *InstFileName* specification is invalid or missing.
`XIPC_ER_BADIDENTFILE`	The instance ".ids" file is inaccessible.
`XIPC_ER_BUSY`	User already connected to an *X•IPC* instance.
`XIPC_ER_GHOSTSTOP`	Cannot deregister with *X•IPC* object daemon.
`XIPC_ER_IDLD`	Cannot communicate with `xipcidld` daemon
`XIPC_ER_INSTNOTACTIVE`	Instance not active.
`XIPC_ER_LOGGEDIN`	User logged in to instance to be stopped.
`XIPC_ER_NOTLOCAL`	XipcStop() was called as part of an *X•IPC* /Network program.
`XIPC_ER_SYSERR`	An internal error has occurred while processing the request.

INTERACTIVE COMMAND

SYNTAX

```
xipcstop  InstName [Options]
```

ARGUMENTS

InstanceName	Name of instance to be stopped: Either an Instance File Name or an instance name (local or network) starting with the character '@'. The instance name can be omitted and specified as '*', in which case the value of the environment variable `XIPC` will be used.
Options	One of: `report`, `force` or `0`.

EXAMPLES

```
xipc> # Use XIPC environment variable to specify instance name.
xipc> # Do not output report.
xipc> xipcstop  * 0
      RetCode = 0
```

3.23 XipcSystemErrorReport() - Get Error Report

NAME

`XipcSystemErrorReport()` - Get an Error Report

SYNTAX

```
#include "xipc.h"


XINT
XipcSystemErrorReport(
          CHAR *Buffer,
          XINT Length);
```

PARAMETERS

None

RETURNS

Value	Description
RC >= 0	Operation succeeded.

RC < 0 Error (see error codes below).

DESCRIPTION

This function will format an error report than can be printed or written to a log file. The information displayed is based on information within the structure XIPC_SYSERR.

The formatted message will have the following format:

```
[System/Network] error reproted by <XIPC Verb>
Operating System: <Operating System>, Failing Service: <Systme Call>
Status1 = <ErrorStatus1>, Status2 = <ErrorStatus2>
Message = <ErrorMessage>
Error Class = <Error Class>: <Error Text>

XIPC call stack:
<Function1>
<Function2>
  .
  .
```

ERRORS

Code	Description
XIPC_ER_LOGINNOTFOUND	User not logged into instance (user never logged in, was aborted or disconnected).
XIPC_ER_BADBUFFER	*MsgBuf* is NULL.
XIPC_ER_MEMORY	Out of system memory.

3.24 XipcTerm() – Terminate the X•IPC Platform Environment

NAME

XipcTerm() – Terminate the X•IPC Platform Environment

SYNTAX

```
#include "xipc.h"

XINT
XipcTerm()
```

PARAMETERS

None

RETURNS

Value	Description
RC >= 0	Instance stopped successfully.
RC < 0	Error (see error codes below).

DESCRIPTION

XipcTerm() is used to terminate the X•IPC environment on a computer platform from under program control. Alternatively, the `xipcterm` command can be executed from the operating system prompt. XipcTerm() closes all internal structures and background processes needed for supporting X•IPC activity on the platform.

Refer to the relevant Platform Notes for platform-specific aspects of XipcTerm().

ERRORS

Code	Description
MEM_ER_FAILSTOP	MemSys termination failed.
MEM_ER_GHOSTSTOP	Cannot deregister MemSys with X•IPC object daemon.
QUE_ER_FAILSTOP	QueSys termination failed.
QUE_ER_GHOSTSTOP	Cannot deregister QueSyswith X•IPC object daemon.
XIPC_ER_BUSY	User already connected to an X•IPC instance.
XIPC_ER_GHOSTSTOP	Cannot deregister with X•IPC object daemon.
XIPC_ER_INSTNOTACTIVE	Instance not active.
XIPC_ER_MEMORY	Out of system memory.
XIPC_ER_NOROOTDIR	Cannot locate X•IPC root directory
XIPC_ER_NOTLOCAL	XipcStop() was called as part of an X•IPC /Network program.
XIPC_ER_PLATNOTACTIVE	Platform configuration file cannot be opened.

XIPC_ER_SYSERR	An internal error has occurred while processing the request.

INTERACTIVE COMMAND

SYNTAX

xipcterm

ARGUMENTS

None

EXAMPLES

```
xipc> xipcterm
      xipcterm:XIPC Platform Environment Terminated
      RetCode = 0
```

3.25 XipcUnfreeze() - Unfreeze Instance

NAME

XipcUnfreeze() - Unfreeze Instance

SYNTAX

```
#include "xipc.h"


XINT
XipcUnfreeze()
```

PARAMETERS

None.

RETURNS

Value	Description
RC >= 0	XipcUnfreeze successful.
RC < 0	Error (see error codes below).

DESCRIPTION

XipcUnfreeze() unfreezes the X•IPC instance. This function provides the same functionality as calling QueUnfreeze(), SemUnfreeze() and MemUnfreeze() as a unit.

For a detailed description of this function, see the description of the subsystem unfreeze functions.

ERRORS

See errors for QueUnfreeze(), SemUnfreeze() and MemUnfreeze().

INTERACTIVE COMMAND

SYNTAX

```
xipcunfreeze
```

ARGUMENTS

None.

EXAMPLES

```
xipc> xipcunfreeze
      RetCode = 0
```

3.26 XipcUnmaskTraps() - Deactivate Trap Mask

NAME

XipcUnmaskTraps() - Deactivate Trap Mask

SYNTAX

```
#include "xipc.h"


XINT
XipcUnmaskTraps()
```

PARAMETERS

None.

RETURNS

Value	Description
RC >= 0	XipcUnmaskTraps successful.
RC < 0	Error (see error codes below).

DESCRIPTION

XipcUnmaskTraps() deactivates the calling user's trap function mask. XipcUnmaskTraps() will fail if the user has not previously masked traps via XipcMaskTraps().

The call to XipcUnmaskTraps() executes trap functions that were prevented from running while the mask was active. Similarly, asynchronous *X•IPC* operations that completed but were not processed (e.g., running their associated callback functions), are processed in the order that they backed up while the mask was active.

Refer to the manual pages for XipcMaskTraps() and XIPC_TRAP_FUNCTION_TEST() for the details of trap masking within *X•IPC* .

ERRORS

Code	Description
XIPC_ER_NOTLOGGEDIN	User not logged into *X•IPC* (User never logged in, is disconnected or was aborted).
XIPC_ER_NOTMASKED	Traps not masked.

INTERACTIVE COMMAND

SYNTAX

```
xipcunmasktraps
```

ARGUMENTS

None.

EXAMPLES

```
xipc> xipcunmasktraps
      RetCode = 0
```

4. X•IPC MACROS

4.1 XIPC_TRAP_FUNCTION_TEST() - Trap Service Function Test

NAME

XIPC_TRAP_FUNCTION_TEST() - Trap Service Function Test

SYNTAX

```
#include "xipc.h"

XIPC_TRAP_FUNCTION_TEST(TrapName, ... )
```

PARAMETERS

Name	Description
TrapName	The name of the trap service function (i.e. a pointer to it).
...	Additional arguments may be required, and are operating system dependent. See the appropriate Platform Notes for their description.

RETURNS

None.

DESCRIPTION

XIPC_TRAP_FUNCTION_TEST(), when placed at the start of a trap service function, prevents the execution of that trap function while the process has its trap mask active. Should a trap occur, the trap function's execution is postponed until the process' trap mask is deactivated.

X•IPC internally activates a calling user's trap mask during internal processing of *X•IPC* functions calls. This is done to guarantee the integrity of *X•IPC*.

A user program can activate and deactivate trap masks directly via the XipcMaskTraps() and XipcUnmaskTraps() function calls.

The call to the XIPC_TRAP_FUNCTION_TEST() macro should generally be the first executable statement in trap service functions that make *X•IPC* function calls–possibly preceded only by the operating system calls required to reset the signal or trap if so desired.

The call to XIPC_TRAP_FUNCTION_TEST() determines if the user's trap mask is active. If it is, then the trap service function is re-scheduled to be executed as soon as the trap mask is deactivated.

Note that statements preceding the call to this macro might be executed a second time if the function is re-scheduled.

ERRORS

None.

5. X•IPC ERROR CODES

The following tables list X•IPC error codes for non-subsystem-specific errors; subsystem error codes will be found in the subsystem documentation.

5.1 By Symbolic Error Name

SYMBOLIC ERROR NAME	NUMBER	DESCRIPTION
XIPC_ER_AMBIGUOUS	-1017	The *Uid* specified as XIPC_USER_NULL but the login cannot be identified uniquely.
XIPC_ER_BADBUFFER	-1019	*MsgBuf* is NULL.
XIPC_ER_BADCONFIGFILE	-1012	The specified *InstFileName* file is inaccessible or contains erroneous contents.
XIPC_ER_BADCONFIGINFO	-1013	The specified *InstFileName* file is inaccessible or contains erroneous contents.
XIPC_ER_BADCONFIGNAME	-1011	The *InstFileName* specification is invalid or missing.
XIPC_ER_BADIDENTFILE	-1025	The instance ".ids" file is inaccessible.
XIPC_ER_BADTEMPFILE	-1026	The instance ".tmp" file is inaccessible.
XIPC_ER_BADUID	-1023	Invalid *Uid* parameter.
XIPC_ER_BADUSERNAME	-1021	Invalid *UserName* parameter.
XIPC_ER_BUSY	-1018	User already connected to an X•IPC instance.
XIPC_ER_CAPACITY_THREAD	-1140	Thread table full.
XIPC_ER_FILEOPEN	-1020	Error opening file.
XIPC_ER_GHOSTSTART	-1084	Cannot register with X•IPC object daemon.
XIPC_ER_GHOSTSTOP	-1085	Cannot deregister with X•IPC object daemon
XIPC_ER_IDLD	-1127	Cannot communicate with `xipcidld` daemon
XIPC_ER_INSTACTIVE	-1081	Instance is already active.
XIPC_ER_INSTNOTACTIVE	-1082	Instance not active.
XIPC_ER_INVALIDSETOPTION	-1124	Invalid Option specified to function.
XIPC_ER_ISMASKED	-1009	Trap mask already active.
XIPC_ER_LOGGEDIN	-1117	User logged in to instance to be stopped.
XIPC_ER_LOGINNOTFOUND	-1016	User not logged into instance (user never logged in, was aborted or disconnected).
XIPC_ER_MEMORY	-1121	Out of system memory
XIPC_ER_NOACCESS	-1001	Unable to connect to X•IPC .
XIPC_ER_NOPLATFORMFILE	-1088	Platform configuration file cannot be opened,
XIPC_ER_NOROOTDIR	-1027	Cannot locate X•IPC root directory..
XIPC_ER_NOTCONNECTED	-1015	User not connected to an instance.
XIPC_ER_NOTFOUND	-1033	Entry not found in the *login working set*.
XIPC_ER_NOTLOCAL	-1037	XipcStart() or XipcStop() was called as part of an X•IPC / Network program.

SYMBOLIC ERROR NAME	NUMBER	DESCRIPTION
XIPC_ER_NOTLOGGEDIN	-1002	User not logged into X•IPC (User never logged in, is disconnected or was aborted).
XIPC_ER_NOTMASKED	-1010	Traps not masked.
XIPC_ER_OPTIONNOTSET	-1125	Option specified has not been set.
XIPC_ER_PLATNOTACTIVE	-1089	Platform configuration file cannot be opened.
XIPC_ER_SYSERR	-1101	An internal error has occurred while executing the operation.
XIPCNET_ER_BADHOST	-2219	Host Name provided is not valid.
XIPCNET_ER_BADNETNAME	-2212	Local or network name missing "@" prefix.
XIPCNET_ER_CONNECTLOST	-2211	Connection to instance lost.
XIPCNET_ER_HOSTUNREACHABLE	-2218	Host may be unreachable.
XIPCNET_ER_NETERR	-2208	Network transmission error
XIPCNET_ER_NOFILE	-2205	Cannot open “XIPCHOSTLIST” or “XIPCCATLIST” file.
XIPCNET_ER_NOFORK	-2206	Process limit exceeded on server.
XIPCNET_ER_NOSEARCH	-2202	Instance search range not defined.
XIPCNET_ER_NOTFOUND	-2201	*InstName* instance not found within instance search range.
XIPCNET_ER_SESSION	-2209	Cannot connect to *InstName* instance.
XIPCNET_ER_SYSERR	-2207	Operating system error.
XIPCNET_ER_TOOBIG	-2210	Message text exceeds instance's size limit.

5.2 By Message Number

NUMBER	SYMBOLIC ERROR NAME	DESCRIPTION
-1001	XIPC_ER_NOACCESS	Unable to connect to X•IPC .
-1002	XIPC_ER_NOTLOGGEDIN	User not logged into X•IPC (User never logged in, is disconnected or was aborted).
-1009	XIPC_ER_ISMASKED	Trap mask already active.
-1010	XIPC_ER_NOTMASKED	Traps not masked.
-1011	XIPC_ER_BADCONFIGNAME	The *InstFileName* specification is invalid or missing.
-1012	XIPC_ER_BADCONFIGFILE	The specified *InstFileName* file is inaccessible or contains erroneous contents.
-1013	XIPC_ER_BADCONFIGINFO	The specified *InstFileName* file is inaccessible or contains erroneous contents.
-1015	XIPC_ER_NOTCONNECTED	User not connected to an instance.
-1016	XIPC_ER_LOGINNOTFOUND	User not logged into instance (user never logged in, was aborted or disconnected).
-1017	XIPC_ER_AMBIGUOUS	The *Uid* specified as XIPC_USER_NULL but the login cannot be identified uniquely.
-1018	XIPC_ER_BUSY	User already connected to an X•IPC instance.
-1019	XIPC_ER_BADBUFFER	*MsgBuf* is NULL.
-1020	XIPC_ER_FILEOPEN	Error opening file.
-1021	XIPC_ER_BADUSERNAME	Invalid *UserName* parameter.
-1023	XIPC_ER_BADUID	Invalid *Uid* parameter.
-1025	XIPC_ER_BADIDENTFILE	The instance ".ids" file is inaccessible.
-1026	XIPC_ER_BADTEMPFILE	The instance ".tmp" file is inaccessible.
-1027	XIPC_ER_NOROOTDIR	Cannot locate X•IPC root directory..
-1033	XIPC_ER_NOTFOUND	Entry not found in the *login working set*.
-1037	XIPC_ER_NOTLOCAL	XipcStart() or XipcStop() was called as part of an X•IPC / Network program.
-1081	XIPC_ER_INSTACTIVE	Instance is already active.
-1082	XIPC_ER_INSTNOTACTIVE	Instance not active.
-1084	XIPC_ER_GHOSTSTART	Cannot register with X•IPC object daemon.
-1085	XIPC_ER_GHOSTSTOP	Cannot deregister with X•IPC object daemon
-1088	XIPC_ER_NOPLATFORMFILE	Platform configuration file cannot be opened,
-1089	XIPC_ER_PLATNOTACTIVE	Platform configuration file cannot be opened.
-1101	XIPC_ER_SYSERR	An internal error has occurred while executing the operation.
-1117	XIPC_ER_LOGGEDIN	User logged in to instance to be stopped.
-1121	XIPC_ER_MEMORY	Out of system memory
-1124	XIPC_ER_INVALIDSETOPTION	Invalid Option specified to function.
-1125	XIPC_ER_OPTIONNOTSET	Option specified has not been set.

NUMBER	SYMBOLIC ERROR NAME	DESCRIPTION
-1127	XIPC_ER_IDLD	Cannot communicate with `xipcidld` daemon
-1140	XIPC_ER_CAPACITY_THREAD	Thread table full.
-2201	XIPCNET_ER_NOTFOUND	*InstName* instance not found within instance search range.
-2202	XIPCNET_ER_NOSEARCH	Instance search range not defined.
-2205	XIPCNET_ER_NOFILE	Cannot open “XIPCHOSTLIST” or “XIPCCATLIST” file.
-2206	XIPCNET_ER_NOFORK	Process limit exceeded on server.
-2207	XIPCNET_ER_SYSERR	Operating system error.
-2208	XIPCNET_ER_NETERR	Network transmission error
-2209	XIPCNET_ER_SESSION	Cannot connect to *InstName* instance.
-2210	XIPCNET_ER_TOOBIG	Message text exceeds instance's size limit.
-2211	XIPCNET_ER_CONNECTLOST	Connection to instance lost.
-2212	XIPCNET_ER_BADNETNAME	Local or network name missing "@" prefix.
-2218	XIPCNET_ER_HOSTUNREACHABLE	Host may be unreachable.
-2219	XIPCNET_ER_BADHOST	Host Name provided is not valid.

6. X•IPC USER DATA STRUCTURES

NAME

X•IPC **General Data Structures** - Data Structures Used by all X•IPC subsystems

SYNTAX

```
/*
 *  The ASYNCRESULT Control Block (ACB) structure is used for examining
 *  the results of an asynchronous operation.  The structure contains
 *  a union that defines returned fields for every XIPC operation
 *  that may block.
 */

/*********************************************************************
**     Macros
*********************************************************************/

#define XIPC_ASYNC_INPROGRESS     1
#define XIPC_ASYNC_COMPLETED      2

#define ACB_FIELD(AcbPtr, Function, Field)      AcbPtr->Api.Function.Field

/*********************************************************************
**     'ACB' - ASYNCRESULT Control Block ---
*********************************************************************/

struct _ASYNCRESULT /* Result of Async API call */
{
       XINT  AUid;                /* Async Uid  "receipt" */
       XINT  AsyncStatus;         /* XIPC_ASYNC_INPROGRESS or XIPC_ASYNC_COMPLETED */
       XINT  UserData1;           /* ------ user defined usage ---- */
       XINT  UserData2;           /* ------ user defined usage ---- */
       XINT  UserData3;           /* ------ user defined usage ---- */

       XINT     OpCode;           /* Async operation, key to union */

       union
       {
              struct
              {
                     XINT         RetSid;
                     XINT         RetCode;      /* of completed async operation */
              }
              SemWait;

              struct
              {
                     XINT         RetSid;
                     XINT         RetCode;      /* of completed async operation */
              }
              SemAcquire;
```

```
struct
{
       MSGHDR MsgHdr;              /* The resultant MsgHdr */
       CHAR FAR      *MsgBuf;
       XINT          RetCode;      /* of completed async operation */
}
QueWrite;




struct
{
       MSGHDR MsgHdr;              /* The resultant MsgHdr */
       XINT          RetQid;
       XINT          RetCode;
}
QuePut;

struct
{
       MSGHDR MsgHdr;              /* The resultant MsgHdr */
       XINT          Priority;
       XINT          RetQid;
       XINT          RetCode;
}
QueGet;

struct
{
       CHAR FAR      *MsgBuf;
       XINT          RetQid;
       XINT          RetCode;
}
QueSend;

struct
{
       CHAR FAR      *MsgBuf;
       XINT          MsgLen;
       XINT          Priority;
       XINT          RetQid;
       XINT          RetCode;
}
QueReceive;

struct
{
       /*
        * Only used for passing error info re
        * failed QueBurstSend() operation.
        */
       XINT          TargetQid;
       CHAR FAR      *MsgBuf;      /* Original text */
       XINT          MsgLength;
       XINT          Priority;
```

```
        XINT          RetQid;
        XINT          RetCode;
}
QueBurstSend;

struct
{
        /*
         * Only used for handling an asynchronous
         * QueBurstSendSync() operation.
         */
        XINT          SeqNo; /* of last burst-send msg enqueued */
        XINT          RetCode;
}
QueBurstSendSync;

struct
{
        XINT          Mid;          /* of target */
        XINT          Offset;       /* of target */
        XINT          Length;       /* of target */
        CHAR FAR      *Buffer;
        XINT          RetCode;
}
MemWrite;




struct
{
        XINT          Mid;          /* of target */
        XINT          Offset;       /* of target */
        XINT          Length;       /* of target */
        CHAR FAR      *Buffer;
        XINT          RetCode;
}
MemRead;

struct
{
        SECTION       RetSec;
        XINT          RetCode;
}
MemSecOwn;

struct
{
        SECTION       RetSec;
        XINT          RetCode;
}
MemLock;

struct
{
        MOM_MSGID     MsgId;
        XINT          RetCode;
```

```
            }
            MomSend;

            struct
            {
                  CHAR FAR      *MsgBuf;
                  XINT          MsgLen;
                  MOM_MSGID     MsgId;
                  XINT          ReplyAppQueue;
                  XINT          RetCode;
                  XINT          TrackingLevel;
            }
            MomReceive;

            struct
            {
                  XINT          RetCode;      /*…of completed async operation*/
            }
            MomEvent;

      }
      Api;

};
```

7. SAMPLE PROGRAMS

This section contains a description of sample and demo programs supplied with the *X•IPC* tool kit. The program source can be found in the `samples` subdirectory of the *X•IPC* installation directory.

7.1 Sample Source Code Listing - `sample.c`

The following is a complete source code listing of the `sample.c` program that is included in the directory of sample programs.

```
/*********************************************************************
**
**  Copyright (C) Envoy Technologies Inc.
**
*********************************************************************/

/*********************************************************************
**
**  Name:           sample.c
**  ----
**
**  Purpose:    XIPC Sample Program
**  -------
**
**  Description:
**  -----------
**      This program creates activity on an XIPC instance.
**      Specifically, it loops a given number of cycles
**      during which it acquires and releases an XIPC resource semaphore,
**      sends or receives a message from an XIPC message queue,
**      and locks, writes, clears, and unlocks an area of
**      an XIPC shared memory segment.
**
**  Parameters:     [Command Line]
**  ----------
**      Cycles      Number of cycles (Integer).
**      SleepVal    Time (in ms) to sleep between steps (Integer).
**      ProdCons    Queue operation type (Char: P or C).
**      FillChar    Character to fill memory with (Char).
**      Instance    Name of XIPC Instance to use (String)
**
**
**  Sample Usage:
**  ------------
**      (NOTE:  At least two programs must be started with
**              one being a Producer and another a Consumer).
**
**      sample 10000 50 P z @testinst
**      sample 10000 50 C x @testinst
**
**
**  Returns:
**  -------
**      0   -   Success
**      1   -   Failure.
**
**
**
**  Note:
**  -----
**      This program does not include provisions for signals.
**
*********************************************************************/
```

```
/************************************************************************
** header files
************************************************************************/

#include <stdio.h>
#include <time.h>
#include "xipc.h"


/************************************************************************
** global constants
************************************************************************/

#define SIZE_SEG 260L
#define SIZE_BUF 512L


/************************************************************************
** functions
************************************************************************/

XINT SemErrorCheck();
XINT QueErrorCheck();
XINT MemErrorCheck();
VOID Terminate();


/************************************************************************
** Main Program
************************************************************************/

XINT
main(ArgC, ArgV)
XINT  ArgC;
CHAR  **ArgV;

{
    /*
     * Declare necessary local stuff.
     */

    CHAR *Instance = NULL;
    CHAR LoginName[16];
    CHAR QueCode, FillChar;
    CHAR Buf[SIZE_BUF];
    MIDLIST MidList;
    QIDLIST QidList;
    SIDLIST SidList;
    SECTION MemSec, TempSec, RetSec;
    XINT Offset, Size, BufLen;
    XINT Cycles, i, j;
    XINT Uid, Mid, Qid, Sid;
    XINT PrioIn, PrioOut;
    XINT RetCode, RetQid, RetSid;
    XINT Pid, SleepVal;
    XINT TimeStart, TimeEnd;
```

```
/*******************************************************
 **
 ** Initialize stuff.
 **
 *******************************************************/

/*
 * Test usage.
 */

if (ArgC < 6)
{
    printf(
    "Usage: sample <Cycles> <SleepVal> [P|C] <FillChar> <Instance>\n");
    exit(1);
}


/*
 * Read command line parameters.
 */

Cycles = atol(ArgV[1]);
SleepVal = atol(ArgV[2]);
QueCode = ArgV[3][0];
FillChar = ArgV[4][0];
Instance = ArgV[5];


/*
 * Create XIPC login name. Also initialize rand().
 */

Pid = getpid();
sprintf(LoginName, "Pgm%05d", Pid);
srand((INT)Pid);


/*
 * Login to target instance.
 */

if ((RetCode = XipcLogin(Instance, LoginName)) < 0)
{
    printf("sample: XipcLogin failed, RetCode = %d\n", RetCode);
    exit(1);
}
Uid = RetCode;
```

```
/*
 * Create resource semaphore having one resource copy.
 * Access it, if already exists. Also build SidList
 * to be used in loop below.
 */

if ((RetCode = SemCreate("SampSem", 1L)) < 0)
{
    RetCode = SemAccess("SampSem");
    SemErrorCheck("SemAcess", RetCode, -1L);
}
Sid = RetCode;
SemListBuild(SidList, Sid, SEM_EOL);


/*
 * Create message queue, having capacity for
 * 40 messages and 4096 bytes. Access it, if
 * already exists. Also build QidList to be
 * used in loop below.
 */

if ((RetCode = QueCreate("SampQue", 40L, 4096L)) < 0)
{
    RetCode = QueAccess("SampQue");
    QueErrorCheck("QueAcess", RetCode, -1L);
}
Qid = RetCode;
QueListBuild(QidList, Qid, QUE_EOL);
PrioOut = 1000L;


/*
 * Create memory segment of SIZE_SEG bytes.
 * Access it, if already exists.
 */

if ((RetCode = MemCreate("SampSeg", SIZE_SEG)) < 0)
{
    RetCode = MemAccess("SampSeg");
    MemErrorCheck("MemAcess", RetCode,
                MemSectionBuild(&TempSec, -1L, 0L, 0L));
}
Mid = RetCode;



/*******************************************************
 **
 ** Run the loop.
 **
 *******************************************************/

TimeStart = time((XINT *) 0);

for (j = 0; j < Cycles; j++)
{
```

```
/*.............SemSys work.....................*/

/*
 * Acquire the resource semaphore.
 */

RetCode = SemAcquire(SEM_ALL, SidList, &RetSid, SEM_WAIT);
SemErrorCheck("SemAcquire", RetCode, RetSid);


/*
 * pause ...
 */

XipcSleep(SleepVal);


/*
 * Release the resource semaphore.
 */

RetCode = SemRelease(SidList, &RetSid);
SemErrorCheck("SemRelease", RetCode, RetSid);


/*.............QueSys work.....................*/

/*
 * Determine if we are a consumer or producer.
 */

switch (QueCode)
{

    case 'P':   /* Producer */
    case 'p':

        /*
         * Send a random size message onto queue.
         */

        BufLen = ((XINT)rand() % SIZE_BUF) + 1;
        RetCode = QueSend(QUE_Q_ANY, QidList, Buf, BufLen,
                          PrioOut, &RetQid, QUE_WAIT);
        QueErrorCheck("QueSend", RetCode, RetQid);
        break;
```

```
    case 'C':   /* Consumer */
    case 'c':

        /*
         * Receive oldest message from queue.
         */

        RetCode = QueReceive(QUE_Q_EA, QidList, Buf,
                        (XINT)sizeof(Buf), &PrioIn,
                        &RetQid, QUE_WAIT);
        QueErrorCheck("QueReceive", RetCode, RetQid);
        break;

}


/*.................MemSys work......................*/


/*
 * Pick a random section to lock and fill.
 * Build MidList containing the section.
 */

Offset = (XINT)rand() % SIZE_SEG;
Size   = ((XINT)rand() % (SIZE_SEG - Offset)) + 1;
MemSectionBuild(&MemSec, Mid, Offset, Size);
MemListBuild(MidList, MemSec, MEM_EOL);


/*
 * Lock the section.
 */

RetCode = MemLock(MEM_ALL, MidList, &RetSec, MEM_WAIT);
MemErrorCheck("MemLock", RetCode, RetSec);


/*
 * Fill with FillChar.
 */

RetCode = MemWrite(Mid, (XINT) Offset, Size, MEM_FILL(FillChar),
            MEM_WAIT);
MemErrorCheck("MemWrite", RetCode, MemSec);


/*
 * pause ...
 */

XipcSleep(SleepVal);
```

```
        /*
         * Clear the section. (i.e., fill with blanks).
         */

        RetCode = MemWrite(Mid, (XINT) Offset, Size, MEM_FILL(' '), MEM_WAIT);
        MemErrorCheck("MemWrite", RetCode, MemSec);


        /*
         * Unlock the section.
         */

        RetCode = MemUnlock(MidList, &RetSec);
        MemErrorCheck("MemUnlock", RetCode, RetSec);
    }


    /*********************************************************
     **
     ** Finish up.
     **
     *********************************************************/

    /*
     * Calculate elapsed time.
     */

    TimeEnd = time((XINT *) 0);
    printf("sample: Time = %ld\n", TimeEnd - TimeStart);



    /*
     * Say goodnight ...
     */

    Terminate(0);


} /* main */
```

```
/*********************************************************************
**
**  Name:   SemErrorCheck
**  ----
**
**  Description:
**  -----------
**      This functions checks for errors during SemSys operations.
**
*********************************************************************/

XINT
SemErrorCheck(FunctionName, RetCode, Sid)
CHAR   *FunctionName;
XINT    RetCode;
XINT    Sid;

{
    if (RetCode < 0)
    {
        fprintf(stderr, "sample: %s failed, RetCode = %d, Sid = %d\n",
                FunctionName, RetCode, Sid);
        Terminate(1);
    }
}


/*********************************************************************
**
**  Name:   QueErrorCheck
**  ----
**
**  Description:
**  -----------
**      This functions checks for errors during QueSys operations.
**
*********************************************************************/

XINT
QueErrorCheck(FunctionName, RetCode, Qid)
CHAR    *FunctionName;
XINT    RetCode;
XINT    Qid;

{
    if (RetCode < 0)
    {
        fprintf(stderr, "sample: %s failed, RetCode = %d, Qid = %d\n",
                FunctionName, RetCode, Qid);
        Terminate(1);
    }
}
```

```
/*********************************************************************
**
**  Name:    MemErrorCheck
**  ----
**
**  Description:
**  -----------
**      This functions checks for errors during MemSys operations.
**
*********************************************************************/

XINT
MemErrorCheck(FunctionName, RetCode, Section)
CHAR    *FunctionName;
XINT    RetCode;
SECTION *Section;

{

    if (RetCode < 0)
    {
        fprintf(stderr,
            "sample: %s failed, RetCode = %d, Section = (%d %ld %ld)\n",
            FunctionName, RetCode, Section->Mid, Section->Offset,
                    Section->Size);
        Terminate(1);
    }
}
```

```
/*********************************************************************
**
**  Name:   Terminate
**  ----
**
**  Description:
**  -----------
**      This functions logs user out from XIPC and terminates.
**
*********************************************************************/

VOID
Terminate(ExitCode)
XINT    ExitCode;

{
    XINT RetCode;

    if ((RetCode = XipcLogout()) < 0)
    {
        printf("\nsample: XipcLogout failed. RetCode = %d.\n", RetCode);
        ExitCode = 1;
    }
    exit(ExitCode);
}

/*
** END ***************************************************************
*/
```

7.2 Other Sample Programs

Additional sample programs are included with the X•IPC tool kit. They demonstrate X•IPC coding methods within a variety of different contexts. The following is a list of some of the included samples.

- `momprod/momcons.c` is a program that illustrate a simple Producer/Consumer pair via synchronous MomSend() and MomReceive() operations.
- `momclnt/momsrvr.c` is a program that illustrate a client and server communicating via MomSys with the `REPLYTO` option to MomSend() and MomReceive(). Though the client maintains an anonymous identity with respect to the server, he will retrieve only responses to the requests he has made.
- `demmsg.c` is program that either generates artificial stock market message traffic on a X•IPC message queue, or absorbs such messages from a queue, depending on a command line option.
- `demqtest.c` is a program that cycles through a series of message send/receive operations using X•IPC message queues.
- `demburst.c` is a QueSys producer program using QueBurst functionality with benchmarking statistics.
- `demwatch.c` is a program that illustrates sophisticated use of MemSys shared-memory segments; fluid movement of data through the X•IPC instance may be viewed with the MemSys real-time monitor.exercises an X•IPC instance's MemSys by means of a series of lock, write, and unlock operations.
- `demmemop.c` is a program that presents an example of extending X•IPC 's functionality. The program demonstrates this capability by building a software function for atomically incrementing an arbitrary word of memory in an X•IPC shared memory segment. This aspect of X•IPC is described in the following section.
- `demrtest.c` is a program that cycles through a series of acquire/release operations using X•IPC resource semaphores.
- `demetest.c` is a program that cycles through a series of wait/set sequences using X•IPC event semaphores.
- `demmix.c` is a program that exercises the SemSys, QueSys and MemSys of an X•IPC instance.
- `sample.c` is a program that uses many areas of X•IPC: sends and receives message to QueSys message queues, acquires and releases SemSys semaphores, and locks and writes MemSys shared-memory segments.
- `qdemo.c` is a program for a multi-threaded QueSys Consumer/Producer.
- `aqdemo.c` is a program for a multi-threaded QueSys Consumer/Producer program using asynchronous I/O.

- `demo_aio.c` is a program that echoes back user's input to console via asynchronous I/O QueReceive().
- `demasync` is a program that waits on SemSys resource semaphores asynchronously.
- `test.cfg` is an instance configuration file with no optional parameters specified. It determines an instance that uses all the default parameter values.
- `parms.cfg` is an instance configuration file which lists every configurable parameter. It may be edited to fit specific resource needs.

7.3 Extending X•IPC 's Functionality

X•IPC provides the developer with the means for extending X•IPC 's capabilities beyond its basic functionality. User-written functions, built upon the X•IPC API, can provide greatly expanded and specialized forms of IPC functionality.

Examples of extending X•IPC 's functionality could include user-written functions that:

- Increment a word of shared memory "atomically."
- Analyze the contents of all the messages on a message queue.
- Collect IPC statistics as part of a user-designed IPC monitoring system. Collected data can be used for display purposes or for dynamic system intervention.
- Capture periodic images of message queue, shared memory contents or event semaphore settings.

7.3.1 INCREMENT SHARED MEMORY WORD ATOMICALLY

Consider, writing a user function that increments a four byte "word" of X•IPC shared memory "atomically." The target memory word is to be identified by *Mid* and *Offset*. The function should return the value of the incremented word.

By masking MemSys traps and then freezing the subsystem, a series of MemSys operations can be issued that are guaranteed to be run as an "atomic" unit, without trap function interruption and without other MemSys user operations executing interwoven within.

This is a basic requirement for coding a user-defined atomic operation that issues multiple X•IPC function calls.

Example:

```
/*
 * MemIncr() --- Version 1.
 */

XINT
MemIncr (Mid, Offset)
XINT Mid;
XINT Offset;
{
    XINT    Data;

    /*
     * Stop everything.
     */

    XipcMaskTraps();
    MemFreeze();

    /*
     * Perform the necessary MemSys operations.
     */

    MemRead (Mid, Offset, (CHAR *)&Data, 4L, MEM_NOWAIT);

    Data ++;

    MemWrite (Mid, Offset, (CHAR *)&Data, 4L, MEM_NOWAIT);

    /*
     * Restart everything.
     */

    MemUnfreeze();
    XipcUnmaskTraps();

    return (Data);
}
```

The above example is sufficient for situations where it is known that the MemRead and MemWrite function calls will always have read/write access to the targeted area.

For situations where this is not the case, a more generalized solution can be built. MemLock and MemUnlock are resorted to if the targeted area is not read/write accessible.

Example:

```
/*
 * MemIncr() --- Version 2.
 */

XINT
MemIncr (Mid, Offset)
XINT Mid;
XINT Offset;
{
    XINT        Data;
    XINT        RC;
```

```
        /*
         * Stop everything.
         */

        XipcMaskTraps();
        MemFreeze();


        /*
         * Attempt without locking.
         */

        RC = MemRead(Mid, Offset, (CHAR *)&Data, 4L, MEM_NOWAIT);

        if (RC == MEM_ER_NOWAIT)
        {
            MemUnfreeze();
            XipcUnmaskTraps();
            return(MemIncrLock(Mid, Offset));
        }

        Data ++;

        RC= MemWrite(Mid, Offset, (CHAR *)&Data, 4L, MEM_NOWAIT);

        if (RC == MEM_ER_NOWAIT)
        {
            MemUnfreeze();
            XipcUnmaskTraps();
            return(MemIncrLock(Mid, Offset));
        };


        /*
         * Restart everything.
         */

        MemUnfreeze();
        XipcUnmaskTraps();
        return (Data);
    }
```

```
/*.....................................................*/
/*
 * MemIncrLock() --- Performs increment operation using
 *                   MemLock and MemUnlock.
 */

XINT
MemIncrLock(Mid, Offset)
XINT Mid;
XINT Offset;
{
    SECTION TempSec, RetSec;
    MIDLIST MidList;
    XINT        Data;

    /*
     * Perform the MemIncr operation
     * using MemLock/MemUnlock to wait
     * for target to become accessible.
     */

    XipcMaskTraps();

    MemListBuild(MidList,
        *MemSection(&TempSec, Mid, Offset, 4L),
        MEM_EOL );

    MemLock (MEM_ALL, MidList, &RetSec, MEM_WAIT);
    MemRead (Mid, Offset, (CHAR *)&Data, 4L, MEM_NOWAIT);

    Data ++;

    MemWrite (Mid, Offset, (CHAR *)&Data, 4L, MEM_NOWAIT);
    MemUnlock (MidList, &RetSec);

    XipcUnmaskTraps();
    return (Data);
}
```

This version will perform like the first example, so long as the calling user has read/write access to the targeted memory area. If the area is found inaccessible by either MemRead or MemWrite then the operation is performed using a memory locking approach by a call to MemIncrLock.

To summarize, the ability to extend *X•IPC*'s functionality greatly broadens the range of IPC application requirements that can be addressed using the *X•IPC* product.