

# The InterOperability Platform (IOP) Manual

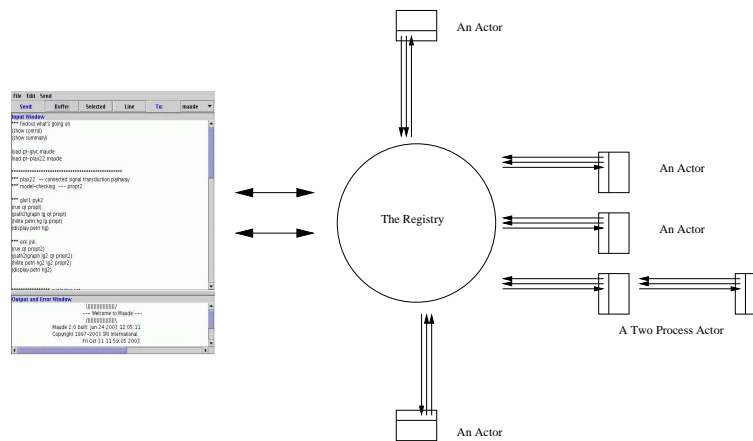
Ian A. Mason\*

University of New England, Armidale, NSW, Australia  
iam@turing.une.edu.au

Carolyn L. Talcott†

SRI, Menlo Park, California, USA  
clt@cs.sri.com

February 16, 2004



\*Most of the work described here was done while holding an International Fellowship at SRI, Menlo Park, partially supported by an Australian Research Council Discovery grant DP0345664, and SRI grant CCR-0234462.

†Partially supported by DARPA through Air Force Research Laboratory Contract F30602-02-C-0130, NSF grants CCR-9900326, CCR-0234462 Office of Naval Research Contract N00014-01-0837.

# Contents

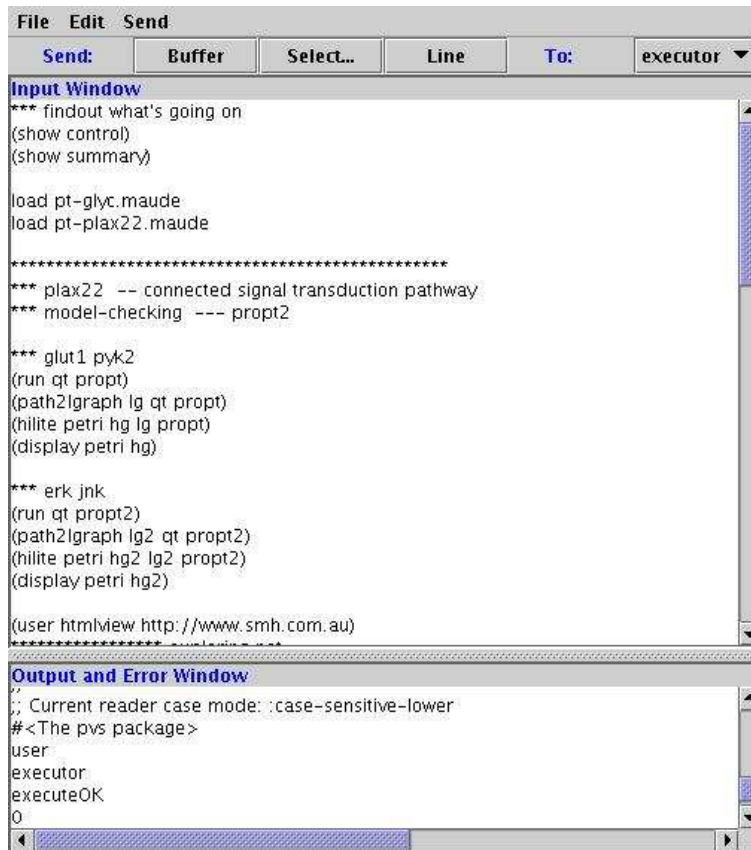
<b>1</b>	<b>Command Line Arguments</b>	<b>3</b>
<b>2</b>	<b>The GUI Front End</b>	<b>3</b>
<b>3</b>	<b>The Actors</b>	<b>3</b>
3.1	The Filemanager Actor . . . . .	3
3.1.1	The Filemanager Read Request . . . . .	4
3.1.2	The Filemanager Write Request . . . . .	4
3.1.3	The Filemanager Append Request . . . . .	5
3.2	The Socketfactory Actor . . . . .	6
3.2.1	The Socketfactory Open Client Request . . . . .	6
3.2.2	The Socketfactory Open Listener Request . . . . .	7
3.2.3	SocketFactory Notes . . . . .	7
3.3	The Socket Actor . . . . .	7
3.3.1	The Socket Read Request . . . . .	8
3.3.2	The Socket Write Request . . . . .	8
3.3.3	The Socket Close Request . . . . .	9
3.4	The Listener Actor . . . . .	10
3.5	The Listener Close Request . . . . .	10
3.5.1	Listener Notes . . . . .	11
3.6	The Executor Actor . . . . .	11
3.7	The Executor Executor Request . . . . .	11
3.7.1	Executor Notes . . . . .	12

## 1 Command Line Arguments

The usual way to start up IOP is via the command:

```
iop <optional maude module>
```

## 2 The GUI Front End



## 3 The Actors

### 3.1 The Filemanager Actor

The Filemanager actor only knows it's request number `<requestNo>`.

### 3.1.1 The Filemanager Read Request

A read request to the Filemanager actor takes the following form:

```
filemanager  
<sender>  
read  
<file>
```

or from the IOP GUI front end:

```
(<sender> read <file>)
```

In response to such a request, the filemanager attempts to open the specified file, lock it, and read its contents. If successful it replies to the <sender> with the appropriate contents. If it fails it logs the reason out to the error logging file and replies with a failure message.

```
<sender>  
filemanager  
contents <file>  
<text>
```

or

```
<sender>  
filemanager  
readFailure  
<file>
```

### 3.1.2 The Filemanager Write Request

A write request to the Filemanager actor takes the following form:

```
filemanager  
<sender>  
write  
<file>  
<text>
```

or from the IOP GUI front end:

```
(<sender> write <file> <text>)
```

In response to such a request, the filemanager attempt to open the file, lock it, and write the supplied text out to the file. The file is created if it doesn't already exist. The previous contents of the file are lost. It would not be difficult to extend the filemanager to have an append mode. If it fails it logs the reason out to the error logging file, and replies with a failure message.

```
filemanager
<sender>
writeOK
<file>
```

or

```
filemanager
<sender>
writeFailure
<file>
```

### **3.1.3 The Filemanager Append Request**

An append request to the Filemanager takes the following form:

```
filemanager
<sender>
append
<file>
<text>
```

or from the IOP GUI front end:

```
(<sender> append <file> <text>)
```

In response to such a request, the filemanager attempts to open the file, lock it, and append the supplied text out to the file. The file is created if it doesn't already exist. If it fails it logs the reason out to the error logging file, and replies with a failure message.

```
filemanager
<sender>
appendOK
<file>
```

or

```
filemanager
<sender>
appendFailure
<file>
```

## 3.2 The Socketfactory Actor

The Socketfactory Actor knows the number of:

- clients it has successfully created, <clientNo>.
- listeners it has successfully created, <listenerNo>.

### 3.2.1 The Socketfactory Open Client Request

An open client request to the Socketfactory actor takes the following form:

```
socketfactory
<sender>
openclient
<host>
<port>
```

or from the IOP GUI front end:

```
(<sender> openclient <host> <port>)
```

In response to such a request, it attempts to connect to the specified <host> and <port>. If successful it creates a new socket actor corresponding to that socket connection. It then replies the the request with the name of the new socket actor. The created actor's name is of the form:

```
"clientsocket<clientNo>"
```

If it is successful it replies with a message of the form:

```
<sender>
socketfactory
openClientOK
<client socket name>
```

Otherwise it replies with:

```
<sender>
socketfactory
openClientFailure
```

### 3.2.2 The Socketfactory Open Listener Request

The open listener Socketfactory request takes the following form:

```
socketfactory  
<sender>  
openlistener  
<port>
```

or from the IOP GUI front end:

```
(<sender> openlistener <port>)
```

In response to such a request, it attempts to create a listening socket on the given port. If successful it creates a new listener actor (with client actor <sender>) that encapsulates that listening socket. The name of the listener actor is of the form:

```
"listener<listenerNo>"
```

If successful:

```
<sender>  
socketfactory  
openListenerOK  
<listener name>
```

Otherwise:

```
<sender>  
socketfactory  
openListenerFailure
```

### 3.2.3 SocketFactory Notes

The SocketFactory actor has a signal handler that waits on any child in response to a SIGCHLD signal delivery. This prevents the exiting of any spawned actors from remaining in the system as zombies.

## 3.3 The Socket Actor

The Socket Actor knows the socket that it corresponds to, and whether or not it is still open. It also keeps track of the number of requests, though this is not used.

### 3.3.1 The Socket Read Request

The Socket read request takes the following form:

```
socket
<sender>
read
<no of bytes>
```

or from the IOP GUI front end:

```
(<sender> read <no of bytes>)
```

In response to such a request, if the socket is still open it attempts to read the specified number of bytes from the socket. This is taken to be an upper limit. If this read is successful (i.e. reads a non-zero number of bytes) it then replies with the number of bytes read, and the bytes read. If it fails either because the socket has been closed, or the read failed, then it logs the reason, and replies with a failure message.

If successful it replies with:

```
<sender>
socket
readOK
<no of bytes read>
<bytes>
```

or

```
<sender>
socket
readFailure
```

otherwise.

### 3.3.2 The Socket Write Request

The Socket write request takes the following form:

```
socket
<sender>
write
<no of bytes>
<bytes>
```

or from the IOP GUI front end:

```
(<sender> write <no of bytes> <bytes>)
```

In response to such a request, if the socket is still open, and <no of bytes> is nonzero, it attempts to write the specified number of bytes to the socket. If this write is successful (i.e. it write <no of bytes> out to the socket) it then replies with the number of bytes written. If it fails either because the socket has been closed, or the write failed, it logs the reason and replies with a failure message. If the <no of bytes> was larger than the number of <bytes> it was supplied with then it writes as much as it can.

If successful it replies with:

```
<sender>  
socket  
writeOK  
<no of bytes read>
```

or

```
<sender>  
socket  
writeFailure
```

otherwise.

### 3.3.3 The Socket Close Request

The Socket close request takes the following form:

```
socket  
<sender>  
close
```

or from the IOP GUI front end:

```
(<sender> close)
```

In response to such a request, if the socket is still open it closes it, and remembers this fact, so subsequent requests will always fail. If the socket is already closed, this like all the other requests will result in a fail reply. Upon closing the actor unregisters with the registry, then exits.

If successful it replies with:

```
<sender>  
socket  
closeOK
```

or

```
<sender>  
socket  
closeFailure
```

otherwise.

### 3.4 The Listener Actor

The Listener Actor knows the listening socket that it is managing. It also knows the number of connections that have been made. A listener actor also knows a client actor `<client>`, the one that requested its creation. It is to this actor that it sends the names of the socket actors it generates per incoming connection. The listener has two threads. One thread monitors the listening socket, the other handles incoming messages.

### 3.5 The Listener Close Request

```
listener  
<sender>  
close
```

or from the IOP GUI front end:

```
(<sender> close)
```

If the listener socket is still open it closes it, and remembers this fact, so subsequent requests will always fail. If the listener is already closed, this will result in a fail reply. After successfully completing this shutdown procedure the actor unregisters with the registry, then exits.

If successful:

```
<sender>  
listener  
closeOK
```

otherwise:

```
<sender>
listener
closeFailure
```

The listening thread does not accept commands.

All it does is listen on it's port, when a connection is made it creates a new socket actor, whose name will be of the form:

```
"connectionsocket.<iop pid>.<requestNo>"
```

and replies:

```
<client>
listener
newConnection
<connection socket name>
```

### 3.5.1 Listener Notes

The Listener actor has a signal handler that waits on any child in response to a SIGCHLD signal delivery. This prevents the exiting of any spawned actors from remaining in the system as zombies.

## 3.6 The Executor Actor

The Executor Actor knows only its request number.

## 3.7 The Executor Executor Request

The Executor executor request takes the following form:

```
executor
<sender>
<command>
```

or from the IOP GUI front end:

```
( <sender> <command> )
```

In response to such a request, the executor forks off a child process, which using the C routine `system( )` executes the command specified by calling `/bin/sh -c <command>`. Once the system call has ended, the child process responds to the `<sender>` with the appropriate exit code, as described by the standard C library.

```
<sender>
executeOK
<exit code>
```

### **3.7.1 Executor Notes**

The Executor actor has a signal handler that waits on any child in response to a SIGCHLD signal delivery. This prevents the exiting of any spawned actors from remaining in the system as zombies.