Introducing XPC

Divide and conquer

Session 206

Damien Sorresso
Architectural Artisan

Introducing XPC

On the agenda

- IPC background
- Designing with XPC
- Using the XPC APIs
- Examples

Interprocess Communication

IPC (According to Wikipedia)

Inter-process communication (IPC) is a set of techniques for the exchange of data among multiple threads in one or more processes.

Interprocess Communication

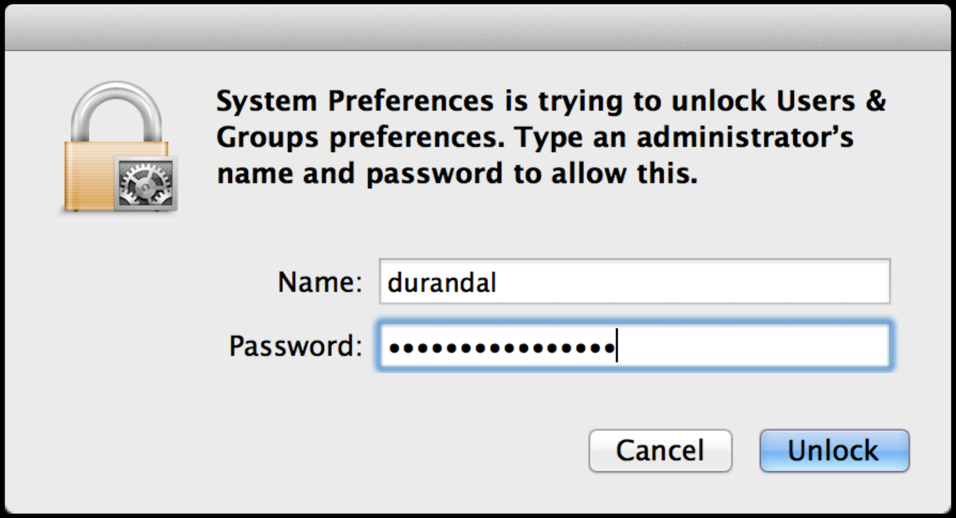
Why use IPC?

m Report
Bomb quit unex
Click Reopen to open t n again. This report will be sent to Apple automatically.
Comments
Show Detail
Reopen

Fault Isolation

Interprocess Communication

Why use IPC?



System Preferences is trying to unlock Users & Groups preferences. Type an administrator's name and password to allow this.

Name: durandal

Password: ••••••••••••••••

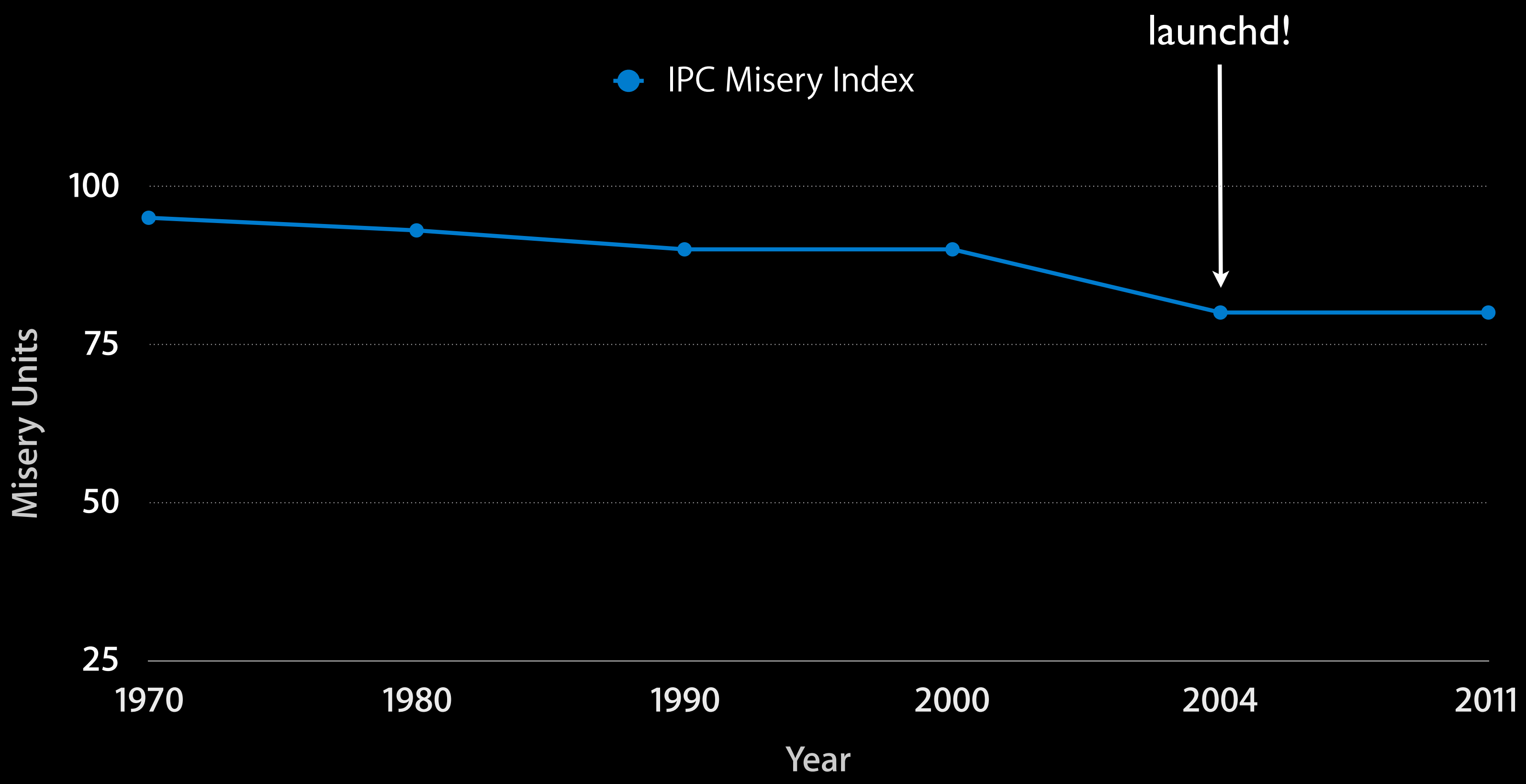
Privilege Separation

Interprocess Communication

POSIX	Mach	Foundation
kill(2)	mach_msg()	NSConnection
signal(3)	MIG	NSProxy
read(2) + write(2)		NSPort
shmat(2)		CFMessagePort
socket(2)		CFMachPort
pipe(2)		

IPC Over Time

Still miserable



Interprocess Communication

1970s and 1980s	launchd	XPC
✓ Setup pipeline	✓ On-demand	✓ Automated bootstrapping
✓ Send bytes		✓ Structured messages

Designing for XPC

Model-View-Controller (with a twist)

Model-View-Controller

- Design pattern to encourage modularity
- Three basic pieces of app functionality separate

Data model **UI presentation** **“Business logic”**

Model-View-Controller

- Encourages modularization at source code level
- Can even be used to modularize at project level (plug-ins)
- But still only applies to one address space

Designing for XPC

Applying MVC concepts

- Isolate various data model pieces into own address spaces
- Apply principle of least privileges to each piece
- OS enforces strict separation of each
- Fundamental redefinition of an app

Designing for XPC

Designing for XPC

Example: QuickTime Player

- Decodes video in sandboxed XPC service
- Uses IOSurface to avoid unnecessary copies
- Crashes in service do not affect QuickTime Player
- Little to no harm if service is exploited

Designing for XPC

Example: Preview

- Sandboxed app
- Uses XPCService to get access to files referenced by PDFs
    - One service parses PDF for file references
    - Other gives Preview access to files
    - Parser does not have filesystem access
- Preview only has access to those files it needs
- Minimizes impact of exploits

Using XPC

Using XPC

Services

Only on Mac OS

New

- Part of your app
- No installation necessary
- Process lifecycle controlled by XPC

XPC Services

In-depth

- Identified by CFBundleIdentifier
- Live in Contents/XPCServices
- Purely on-demand and stateless
- Automatic activity tracking and idle-exit
- Only way to separate privileges in App Sandbox
- Code signing required

XPC Services

Bundle structure

- Photo Uploader.app
  - Contents
    - XPCServices
      - com.mycompany.PhotoUploader.uploader.xpc
      - com.mycompany.PhotoUploader.unzipper.xpc

XPC Services

Bundle structure

- com.mycompany.PhotoUploader.unzipper.xpc
  - com.mycompany.PhotoUploader.unzipper
  - Info.plist
  - Resources

XPC Services

Default environment

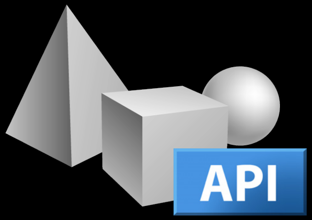
- Restrictive default environment
- Equivalent to background agents (`LSUIElement`)
- Uses GCD run loop (call to `dispatch_main()`)
- Does not share host app’s Keychain access or credentials

XPC Service Info.plist

XPCService dictionary

Key	Value	Description
RunLoop	String	“dispatch_main” (default) “NSRunLoop”
JoinExistingSession	Boolean	If true, joins host app’s session
EnvironmentVariables	Dictionary	Key/value pairs for environment variables

The XPC API



Object API

Transport API

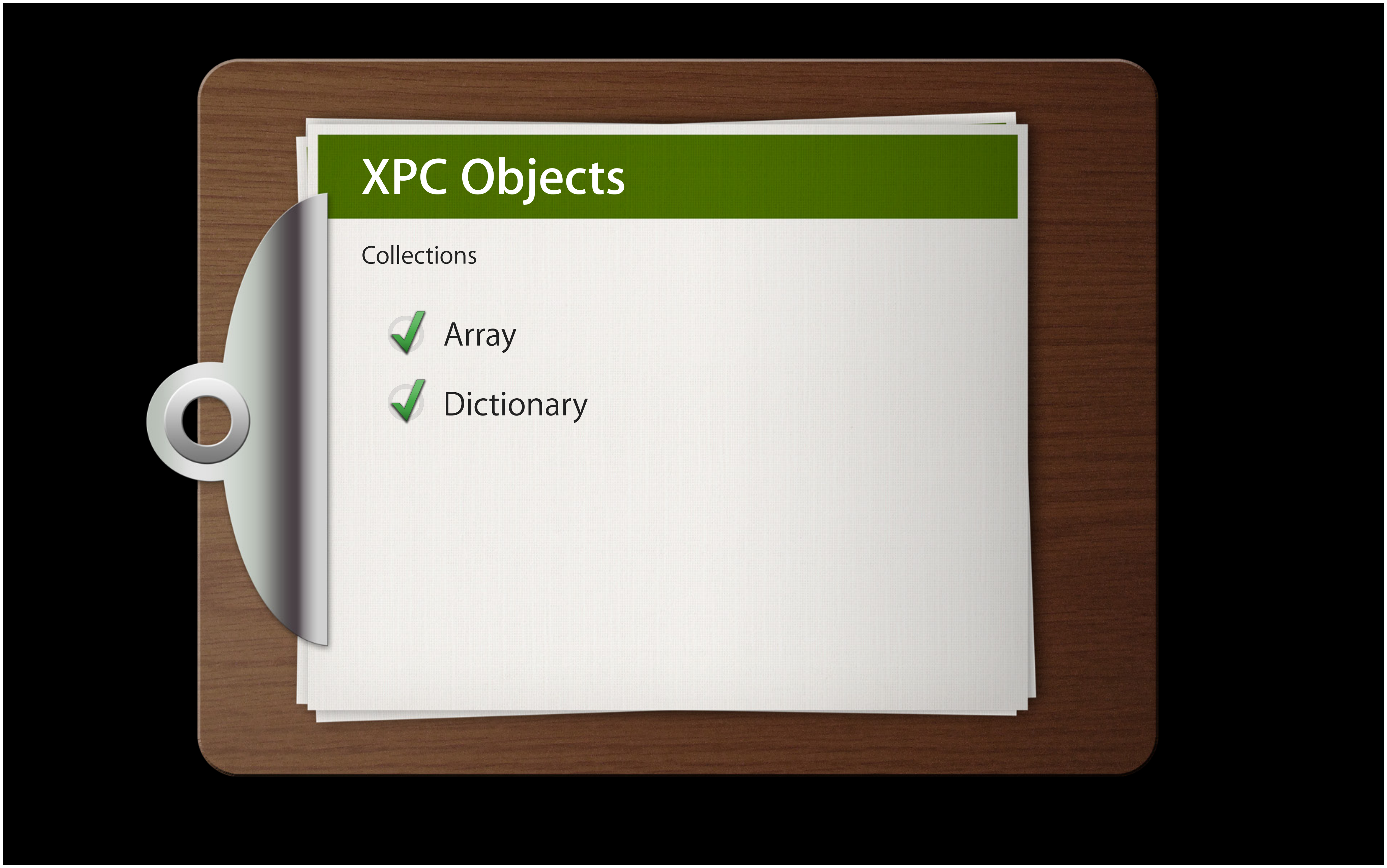
Using XPC

The XPC API

- Object and transport layers unified
- Transport layer understands object layer and vice-versa
- Serialization is an implementation detail

XPC Objects

- Property list-style objects
  - Mutable containers
  - Immutable leaves
- Optimized for packing and unpacking
- Retainable/releasable



XPC Objects

Collections

- ✓ Array
- ✓ Dictionary

XPC Objects

Pure Data

- ✓ Null
- ✓ Boolean
- ✓ Signed Integer
- ✓ Unsigned Integer
- ✓ Double
- ✓ Date
- ✓ Data
- ✓ String
- ✓ UUID

XPC Objects

Out-of-Line

- ✓ File Descriptor
- ✓ Shared Memory
- ✓ IOSurface

XPC Objects

Convenient Container API

- Setters and getters for primitive types
- Allow quick construction/decomposition of messages
- No type-checking needed

XPC Objects

Accessor Defaults

Boolean	→	false
Signed Integer	→	0
Unsigned Integer	→	0
Double	→	NaN
Date	→	0
Data	→	NULL
String	→	NULL
UUID	→	NULL
File Descriptor	→	-1

XPC Objects

```
#include <xpc/xpc.h>

xpc_object_t dictionary, x, y;

dictionary = xpc_dictionary_create(NULL, NULL, 0);

x = xpc_int64_create(640);
y = xpc_int64_create(480);

xpc_dictionary_set_value(dictionary, “X”, x);
xpc_dictionary_set_value(dictionary, “Y”, y);

xpc_release(x);
xpc_release(y);
```

XPC Objects

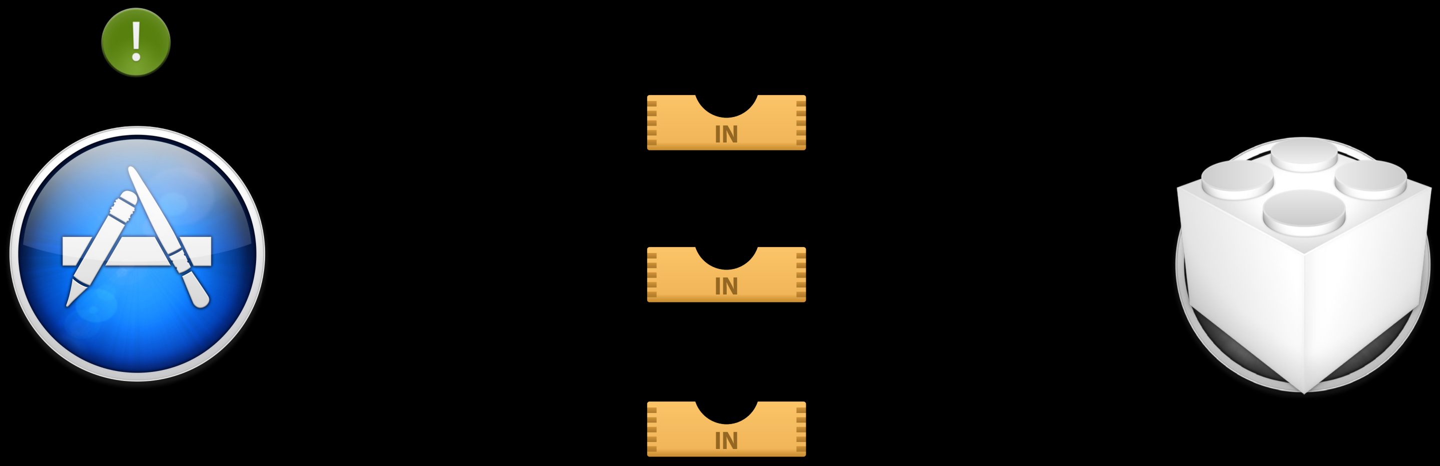
```
xpc_object_t dictionary;

dictionary = xpc_dictionary_create(NULL, NULL, 0);

xpc_dictionary_set_int64(dictionary, “X”, 640);
xpc_dictionary_set_int64(dictionary, “Y”, 480);

int64_t x, y, z;

x = xpc_dictionary_get_int64(dictionary, “X”);
y = xpc_dictionary_get_int64(dictionary, “Y”);
z = xpc_dictionary_get_int64(dictionary, “Z”);
assert(z == 0);
```

XPC Connections

- Virtual—launched on-demand when message is sent
- Bi-directional—allow sending and receiving messages
- Asynchronous/non-blocking—FIFO message delivery

XPC Connections

XPC Connections



XPC Connections

Client-side

- Create connection using service CFBundleIdentifier

```
xpc_connection_t c = xpc_connection_create("com.apple.service", NULL);
xpc_connection_set_event_handler(c, ^(xpc_object_t event) {
    // Always set an event handler. More on this later.
});
xpc_connection_resume(c);

// Messages are always dictionaries.
xpc_dictionary_t message = xpc_dictionary_create(NULL, NULL, 0);
xpc_dictionary_set_uint64(message, "X", 640);
xpc_connection_send_message(c, message);
xpc_release(message)
```

- Each call to `xpc_connection_create()` creates a new peer
- Each peer is distinct

XPC Connections

Client-side

- Message sends are non-blocking
- XPC runtime maintains queue of messages to send
- Use barriers to know when a message is sent

```
xpc_connection_send_message(c, message);
xpc_connection_send_barrier(c, ^{
    // Block is invoked on connection’s target queue
    // when ‘message’ has been sent.
});
xpc_release(message)
```

XPC Connections

Service-side

- Service calls `xpc_main()` with event handler argument
- Event handler receives new peer connections

```
static void
new_connection_handler(xpc_connection_t peer)
{
    xpc_connection_set_event_handler(peer, ^(xpc_object_t event) {
        peer_event_handler(peer, event);
    });
    xpc_connection_resume(xpc_retain(peer));
}

int
main(int argc, const char *argv[])
{
    xpc_main(new_connection_handler);
    exit(EXIT_FAILURE);
}
```

XPC Connections

Service-side (continued)

```
static void
peer_event_handler(xpc_connection_t peer, xpc_object_t event)
{
    if (xpc_get_type(event) == XPC_TYPE_DICTIONARY) {
        // Decompose and handle message.
    } else {
        if (event == XPC_ERROR_CONNECTION_INVALID) {
            // Error indicates the peer has closed the connection.
            // Tear down any associated data structures.
        } else {
            // Error indicates that service will terminate soon.
            // Flush all buffers, finish all work, etc.
        }
        xpc_release(peer);
    }
}
```

XPC Connections

Request-reply

- One-to-one mapping of message to reply block
- Independent of connection's event handler

```
xpc_connection_send_message_with_reply(c, message, q, ^(xpc_object_t reply) {
    if (xpc_get_type(event) == XPC_TYPE_DICTIONARY) {
        // Deconstruct and handle reply.
    } else {
        // Error indicates the service will not reply to the
        // message. Tear down any data structures associated with
        // waiting for the reply.
    }
});
```

XPC Connections

Request-reply (server side)

- Recognizing that a message expects reply is expressed in protocol
- Sending a reply is same as sending normal message

```
static void
peer_event_handler(xpc_object_t event)
{
    xpc_connection_t remote = NULL;
    if (xpc_get_type(event) == XPC_TYPE_DICTIONARY) {
        remote = xpc_dictionary_get_remote_connection(event);

        xpc_object_t reply = xpc_dictionary_create_reply(event);
        xpc_dictionary_set_bool(reply, “reply”, true);

        xpc_connection_send_message(remote, reply);
        xpc_release(reply);
    }
}
```

XPC Connections

Errors

`XPC_ERROR_CONNECTION_INTERRUPTED`

- Re-sync state to other end if needed

`XPC_ERROR_CONNECTION_INVALID`

- Connection no longer useable

`XPC_ERROR_TERMINATION_IMMINENT`

- Prepare to exit cleanly

XPC Connections

Errors in-depth

Error	Connection	Indicates
`XPC_ERROR_CONNECTION_INTERRUPTED`	Peer from `xpc_connection_create()`	Remote end closed connection Connection still usable (Blip in pipeline)
`XPC_ERROR_CONNECTION_INVALID`	Peer received from `xpc_main()` handler	Remote end closed connection Connection unusable
`XPC_ERROR_TERMINATION_IMMINENT`	Peer received from `xpc_main()` handler	Process needs to exit Work still must be finished Flush all buffers

XPC and launchd

Working together to create an on-demand world

XPC and launchd

launchd services

- XPC can be used to talk to launchd jobs
- Use `xpc_connection_create_mach_service()`
- MachServices must be advertised in launchd.plist
- Cannot dynamically register services
- Must manually set up listener
- More complex error cases

XPC and launchd

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://
www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>com.apple.xpc.example</string>
    <key>Program</key>
    <string>/usr/libexec/example</string>
    <key>MachServices</key>
    <dict>
        <key>com.apple.xpc.example</key>
        <true/>
    </dict>
    <key>EnableTransactions</key>
    <true/>
</dict>
</plist>
```

XPC Services

launchd services (client-side)

```
xpc_connection_t listener =
    xpc_connection_create_mach_service(“com.apple.xpc.example”, NULL, 0);

xpc_connection_set_event_handler(listener, ^(xpc_object_t event) {
    // Same semantics as a connection created through
    // xpc_connection_create().
});
xpc_connection_resume(listener);

// Can now send messages.
```

XPC Services

launchd services (server-side)

```
xpc_connection_t listener =
    xpc_connection_create_mach_service(“com.apple.xpc.example”, NULL,
        XPC_CONNECTION_MACH_SERVICE_LISTENER);

xpc_connection_set_event_handler(listener, ^(xpc_object_t event) {
    // New connections arrive here. You may safely cast to
    // xpc_connection_t. You will never receive messages here.
    // The semantics of this handler are similar to those of
    // of the one given to xpc_main().
    new_peer_event_handler((xpc_connection_t)event);
});
xpc_connection_resume(listener);
```

XPC Events

Messages from the system

- Alternate sources of demand
- Arbitrary userspace and kernel events
- Elegant event delivery API
- Enumerate interested events in launchd.plist

XPC Events

IOKit

```
<key>LaunchEvents</key>
<dict>
    <key>com.apple.iokit.matching</key>
    <dict>
        <key>com.apple.device-attach</key>
        <dict>
            <key>idProduct</key>
            <integer>2794</integer>
            <key>idVendor</key>
            <integer>725</integer>
            <key>IOProviderClass</key>
            <string>IOUSBDevice</string>
            <key>IOMatchStream</key>
            <true/>
        </dict>
    </dict>
</dict>
```

XPC Events

BSD Notifications

```
<key>LaunchEvents</key>
<dict>
    <key>com.apple.notifyd.matching</key>
    <dict>
        <key>com.apple.interesting-notification</key>
        <dict>
            <key>Notification</key>
            <string>com.apple.interesting-notification</string>
        </dict>
    </dict>
</dict>
```

XPC Events

Consuming events

- Events received as XPC objects through handler
- If not running, job will be launched on-demand
- Will continue to receive events while running

```
xpc_set_event_stream_handler("com.apple.iokit.matching", q, ^(xpc_object_t event) {
    // Every event has the key XPC_EVENT_KEY_NAME set to a string that
    // is the name you gave the event in your launchd.plist.
    const char *name = xpc_dictionary_get_string(event, XPC_EVENT_KEY_NAME);

    // IOKit events have the IORegistryEntryNumber as a payload.
    uint64_t id = xpc_dictionary_get_uint64(event, "IOMatchLaunchServiceID");

    // Reconstruct the node you were interested in here using the IOKit
    // APIs.
});
```

XPC Events

Consuming events

- Different event streams have different payloads
- Currently support IOKit and BSD Notifications
- More event streams will be added as time goes on

Related Sessions

Introducing App Sandbox	Nob Hill Tuesday, 2:00PM
Blocks and Grand Central Dispatch in Practice	Pacific Heights Wednesday, 10:15AM
Mastering Grand Central Dispatch	Pacific Heights Thursday, 10:15AM

Launch-on-Demand (WWDC2010)	Available on iTunes

Labs

XPC Lab	Core OS Lab B Wednesday, 4:30PM

Documentation

- xpc(3)
- /usr/include/xpc
- devforums.apple.com

WWDC2011