Xgrid Programming Guide

Mac OS X Server > High Performance Computing

2007-10-31

Contents

Introduction **Introduction 7**

Organization of This Document 7

Chapter 1 **Xgrid Overview 9**

How It Works 9
Client Software 11
Controller Software 11
Agent Software 11
Setting Up Xgrid 12
Submitting Jobs to Xgrid 13

Chapter 2 **Getting Started with Xgrid 15**

Before You Start 15
Three Tasks With Different Requirements 16
The Recommended Development Process 16

Chapter 3 **Using the Xgrid Command-Line Client 17**

Basic xgrid Syntax 17
Running a Job Synchronously 18
Submitting a Job for Asynchronous Execution 18
Submitting a Batch Job 19

Chapter 4 **Building and Running GridSample 21**

The GridSample Targets 21
The Xgrid Sample Target 22
The GridFeeder Target 24
Debugging and Monitoring Progress 24

Chapter 5 **Overriding the Job Specification Function 27**

How the GridSample Code Is Organized 27
Changing the Job Specification 28
  Step One: Create a New Job Submission UI 29
  Step Two: Add the Support Code 30
  Step Three: Create an Application Delegate 31
  Step Four: Change the Application Delegate's Class in MainMenu.nib 32
  Summary 33

Chapter 6 Writing a Cocoa Xgrid Client 35

Writing an Application in Six Steps 35
Step One: Locate a Controller and Connect 36
Step Two: Authenticate 36
Step Three: Submit a Job 36
Step Four: Retrieve Job ID 37
Step Five: Register for Notifications 37
Step Six: Data Collection 37
Good Housekeeping 37

Document Revision History 39

Figures and Listings

Chapter 1 **Xgrid Overview 9**

Figure 1-1 Xgrid architecture 10
Figure 1-2 Agent configuration 12

Chapter 3 **Using the Xgrid Command-Line Client 17**

Listing 3-1 Excerpt from the xgrid man page 17
Listing 3-2 Running a job synchronously 18
Listing 3-3 Submitting a job asynchronously 18

Chapter 4 **Building and Running GridSample 21**

Figure 4-1 GridSample project 22
Figure 4-2 The Xgrid Sample controller selection window 23
Figure 4-3 Xgrid Sample main window 23
Figure 4-4 Gridfeeder Sample New Job window 24

Chapter 5 **Overriding the Job Specification Function 27**

Figure 5-1 Locating the right `NewJob.nib` file 29
Figure 5-2 GridSample job submission `.nib` file 30
Figure 5-3 GridCalendar job submission `.nib` file 30
Figure 5-4 GridSampleNewJobWindowContoller files 31
Figure 5-5 Classes tab in `MainMenu.nib` 32
Figure 5-6 `GridCalendarApplicationDelegate` subclass 32
Listing 5-1 GridCalendarApplicationDelegate 31

Introduction

This document describes the programming interfaces to Xgrid, Apple's technology for distributed multiprocessing using multiple computers and multiple processors.

If you are writing an application that can benefit from being executed on multiple processors simultaneously, you should read this document.

Organization of This Document

This document is divided into the following chapters:

- "Xgrid Overview" (page 9)—A description of the Xgrid architecture and tools.
- "Getting Started with Xgrid" (page 15)—Considerations when planning a project for Xgrid, finding the source code, executables, and documentation.
- "Using the Xgrid Command-Line Client" (page 17)—A description of the Apple-supplied `xgrid` command-line client and how to use it.
- "Building and Using GridSample" (page 21)—How to build and use the GridSample Xcode project client application.
- "Overriding the Job Specification Function" (page 27)—How to modify and customize GridSample to create a new application.
- "Writing a Cocoa Xgrid Client" (page 35)—How to write an Xgrid client application from scratch in Objective-C using the Xgrid Foundation framework.

Xgrid Overview

Xgrid allows you to execute programs using multiple computers—and multiple processors on a single computer—to perform multiple calculations in parallel.

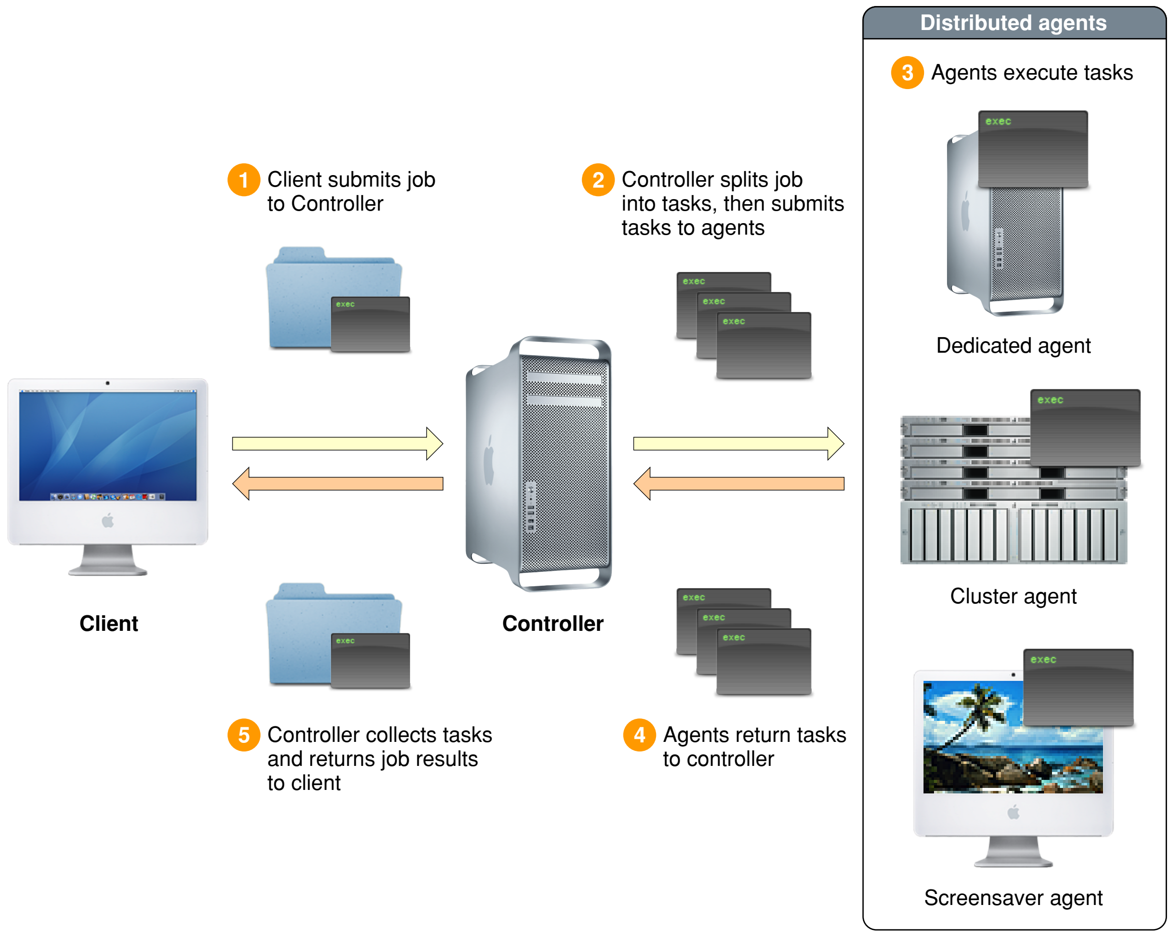
Xgrid is a generalized system, capable of assembling clusters of processors on demand, detecting and correcting failures, and parceling out parallel tasks as needed for general-purpose parallel computing on multiple systems.

The Xgrid controller is a feature of Mac OS X Server, but any computer with Mac OS X (version 10.4 and later) can submit jobs to Xgrid or act as an agent to carry out Xgrid computations.

How It Works

The main components of Xgrid are the *client*, the *controller*, and one or more *agents*. As illustrated in Figure 1-1, the client submits *jobs* to the controller, which assigns *tasks* to the agents. The agents carry out the tasks and return data to the controller. The controller supervises the agents, collects the data, and notifies the client when the job terminates.

Figure 1-1 Xgrid architecture


Distributed agents
3 Agents execute tasks
exec
Dedicated agent
Cluster agent
Screensaver agent
1 Client submits job to Controller
2 Controller splits job into tasks, then submits tasks to agents
Client
Controller
5 Controller collects tasks and returns job results to client
4 Agents return tasks to controller

A *job*, as defined for Xgrid, is a collection of one or more executable *tasks* that can be run in parallel. Each individual task consists of an executable file and any necessary input parameters, data files, and directories.

If enough agents are available, each task is assigned to an agent for simultaneous execution. If necessary, agents with multiple CPUs or multiple cores will have a separate task assigned to each CPU or core.

If there are not enough agents, CPUs, or cores to execute all the tasks simultaneously, the controller assigns tasks to each agent, then waits and assigns the remaining tasks to agents as they finish their current task or otherwise become available.

The controller passes or copies the executable files to the agents, along with any necessary working directories for input and output. The controller supervises and coordinates the agents, detects individual failures, and reassigns tasks as necessary. The agents complete the tasks and return any data, and the controller notifies the client when the job is done or aborts due to an error.

Since jobs typically take a long time to execute, the process is asynchronous. The client submits the job and is notified when it completes.

The client can register to be notified by the controller when the job completes or when events such as errors occur. Notifications by email are also supported.

The client may also monitor the job state at any time by querying the controller. The client may disconnect from the network and return to check the status of the job later.

Xgrid provides notification of errors, task completion, and job completion. Ongoing progress of individual tasks is not reported, even for very long tasks.

Client Software

Mac OS X includes a command-line client, `xgrid`, and a sample client application, GridSample.

The `xgrid` command-line client is installed on all computers with Mac OS X, versions 10.4 and later.

When you install the Developer Tools for Mac OS X version 10.4, or the Xcode Tools for Mac OS X version 10.5, a directory named `Developer/Examples/Xgrid` is created. Inside this directory is an Xcode project named GridSample. This project contains a complete client application. You can build and run this application as a graphical alternative to the `xgrid` command-line tool. You can also modify the GridSample code to create your own client applications with minimal programming. To an extent, you can treat the GridSample project as an application framework.

You can also write your own Xgrid client software from scratch, using the Xgrid Foundation framework—a collection of Objective-C classes.

Client software that you write, either from scratch or by modifying the GridSample code, can be run on any computer with Mac OS X version 10.4 or later.

Controller Software

The controller software is included on Mac OS X Server version 10.4 or later.

You do not normally need to interact with the controller software directly. After a controller is configured, it waits for job submissions from clients and performs its work without human intervention. On Mac OS X Server, the Xgrid controller can be configured using the Server Admin tool (found in the `Applications/Server/` directory). From within Server Admin, choose Computers and Services, then select Xgrid. Tabs are available for configuring controller software and agent software.

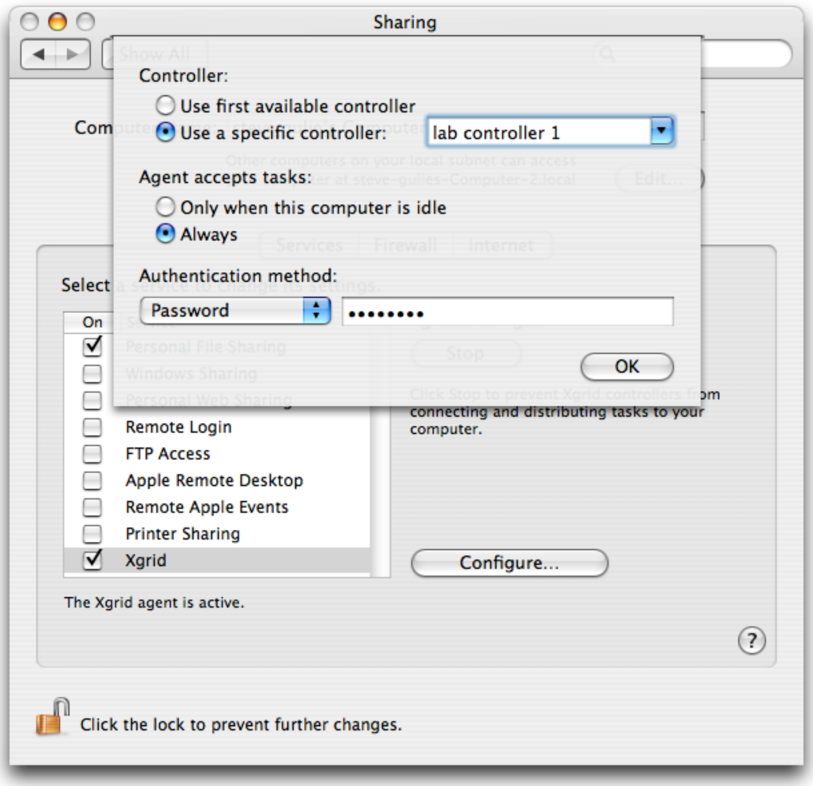
Mac OS X Server also includes the XgridAdmin application (also found in the `Applications/Server` directory), which can be used to monitor and administer Xgrid, to cancel or delete jobs, for example.

Agent Software

Any computer with Mac OS X version 10.3 or later can act as an agent. The agent software is included in version 10.4 and later, and can be downloaded for version 10.3.

The agent software can be controlled from the System Preferences window, as shown in Figure 1-2.

Figure 1-2 Agent configuration



Controller:
( ) Use first available controller
(●) Use a specific controller: lab controller 1

Agent accepts tasks:
( ) Only when this computer is idle
(●) Always

Authentication method:
Password ••••••••

On	Service
✓	Personal File Sharing
☐	Windows Sharing
☐	Personal Web Sharing
☐	Remote Login
☐	FTP Access
☐	Apple Remote Desktop
☐	Remote Apple Events
☐	Printer Sharing
✓	Xgrid

Click Stop to prevent Xgrid controllers from connecting and distributing tasks to your computer.

The Xgrid agent is active.

Click the lock to prevent further changes.

By default, agent software is off. When turned on, by default it accepts tasks only when the host computer is idle (has had no activity for 15 minutes or is running the screen saver). The agent software can be set to accept jobs at any time, however, making the host computer a *dedicated* agent (the host computer can still run other software, but is available for parallel computing tasks at all times).

On Mac OS X Server, the agent softwarer can be configured using the Server Admin tool (found in the `Applications/Server/` directory). From within Server Admin, choose Computers and Services, then select Xgrid. Tabs are available for configuring controller software and agent software.

Setting Up Xgrid

Setting up Xgrid is fairly simple.

- Choose a Mac OS X Server computer to act as a controller. Enable the controller and set the password using the Server Admin application (choose Xgrid from the list of Computers and Services; there are tabs for configuring the Agent and Controller services).
- Choose the computers you will use as agents and enable the Xgrid function (in the Sharing pane of the System Preferences window) for each agent computer.
- Make sure that all of the agents can be accessed over the network by the controller.

- If your agents are sharing access to a common pool of data, rather than receiving copies of all necessary data from the controller, make sure that all the agents have network access to the data and verify that they have the necessary permissions to read and write.

While simple in concept, ensuring network access to all agents in a large organization can be tricky to implement. Similarly, setting permissions correctly so that a data set can be shared by multiple agents can involve a great deal of housekeeping. Where doing so is practical, you can eliminate most of the complexity by taking two simplifying steps: put the controller and all the agents on the same IP subnet; and have the controller copy all necessary data to the agents.

Setting up the controller and agents is described in more detail in [Xgrid Administration Guide]. The administrator's guide also provides a more detailed overview of the Xgrid architecture and setup considerations.

Submitting Jobs to Xgrid

Mac OS X includes two client applications for submitting jobs to Xgrid: a command-line tool named `xgrid`, and an application named GridSample, which is provided as build-able source code. Mac OS X (version 10.4 and later) also contains an Objective-C framework, Xgrid Foundation, which includes a client API for Xgrid. Consequently, there are four ways to submit jobs to Xgrid:

- You can use the `xgrid` command-line client with a headless application, such as a shell script, to submit jobs to Xgrid. This is a utilitarian approach that allows you to use Xgrid in a UNIX-like way without writing any Xgrid-specific code at all.
- You can use the GridSample application in much the way that you use the `xgrid` command-line tool. GridSample has a user interface that is more flexible and approachable than the `xgrid` command, allowing your software to be run by users who may not be comfortable or familiar with the UNIX command line or Terminal interface, without requiring you to write any UI code.
- You can modify the GridSample source code, overriding the job specification function and adding your own nib file to create a customized user interface. Modifying GridSample enables you to quickly create a customized client with its own user interface without writing any unnecessary code. All the details of job submission, monitoring, and I/O handling are built in. This is the recommended approach for university environments or in-house computing projects that do not require an extensively "branded" client application. If you are planning to write a stand-alone application, you should probably modify GridSample as a first step to test the functional part of your code prior to debugging the UI and housekeeping parts.
- You can write a stand-alone Objective-C application that acts as an Xgrid client and submits jobs to Xgrid, using the Xgrid Foundation framework. This is the recommended approach when you need complete control over the look and feel of the Xgrid client application.

This document describes all four methods of submitting jobs to Xgrid.

Getting Started with Xgrid

Writing a client application for Xgrid involves significant planning prior to writing your code. This chapter describes some things to consider when planning a project for Xgrid and shows you how to get started.

Before You Start

The first task is to assess whether your job is suitable for parallel processing with Xgrid. If a job can be broken into a series of independent tasks which can be performed in any order, or with simple order dependencies, it is generally suitable. If complex interdependencies exist between one computational task and another, or the tasks must be performed in linear order, it is generally not suitable.

You can specify minimal task dependencies when you submit a job. For example, you can specify that task D may not begin until tasks A, B, and C are complete. More complex dependencies, however, are better suited to a multiprocessing language such as MPI than to Xgrid.

Note: Xgrid does no logical dependency checking. If you specify that task A must complete before task B is initiated, and vice versa, the job hangs, with each task enqueued and waiting for the other to complete.

The size of the data set being operated on also matters. If a great deal of processing is done on a small data set, the job is better suited to Xgrid than if a small amount of processing is done on a large data set—transferring the data may take more time than is saved by dividing the processing among multiple computers.

It is the responsibility of the client to break the job up into independently executable tasks, and to assemble the collection of executable files, as well as input and output files or directories, into a job submission. You'll find the details of a job submission later in this document, but know for now that breaking the job into executable tasks—with any necessary files and directories—is part of the process.

The next step in the planning process is to assess what type of Xgrid is needed to perform the job in a reasonable amount of time. Some types of jobs can be performed well by a network of loosely connected computers; other jobs require shared access to network file servers, or even dedicated clusters sharing FDDI access to RAID arrays for reasonable efficiency.

The two most important factors in determining the type of Xgrid you need are the size the of the data set being operated on and the amount of processing to be done on the data set. For example, if you are doing a great deal of processing on a relatively small data set, the agents can be loosely connected—by Ethernet, Airport, or even the Internet. If a great deal of data must be processed by a relatively short algorithm, the time spent transferring the data may be greater than the time saved by dividing up the processing, unless shared access to data—or very fast data connections—are available.

Three Tasks With Different Requirements

A common task for Xgrid is video processing. Consider three kinds of job: compressing a short video to several bandwidths for Internet distribution, applying a filter to a long video, and compressing a large video for DVD in three formats: standard television, widescreen, and high definition.

Compressing a short video to several bandwidths can be accomplished simply over Ethernet, and may be practical even with Airport networking. Transferring the video to the agents takes only seconds or a few minutes, while the compression may take many minutes or an hour. Thus the job scales well with an agent assigned to compressing each bandwidth. The returned data is compressed, making its transfer somewhat more efficient.

Applying a simple filter to a long video, however, may require dedicated hardware to benefit from parallel processing at all. In principal, it is easy to divide the job into a parallel set of tasks: simply divide the frames by the number of agents and set each agent to process a set of frames. It may require more time to transfer each frame to and from the agent than it does to apply the filter, however, thus negating any time saving unless the agents share rapid access to the data via shared FDDI access to a RAID or a similar technology.

Compressing a long video to three data-intensive formats falls between these two extremes. The processing may take several hours for each format, so the time saved by parceling out the task to multiple agents is significant (a separate agent for compressing the audio may also save significant time), but the data transfer time is also significant: it may take hours just to send three copies of the video over the same Ethernet backbone. In this case, even though time can probably be saved by using a loosely connected set of agents, fast Ethernet connection is a minimum requirement for reasonable efficiency, and FDDI or a dedicated cluster sharing a RAID will deliver proportionately faster results.

The Recommended Development Process

If you have not already done so, install the Developer Tools or Xcode Tools that came with your copy of Mac OS X and locate the Xgrid folder (in the `Developer/Examples/` folder). Locate the GridSample and GridMandelbrot sample projects.

Before you begin coding, you should use the `xgrid` command-line tool, then compile and run the GridSample application to get a feel for how Xgrid works. See "Using the xgrid Command Line Client" (page 17), and "Building and Running GridSample" (page 21).

When you are ready to begin coding, start by creating a collection of executable files and submitting them as a job using the `xgrid` command line client. When your job is submitting and running properly, continue your code development by modifying the GridSample code, overriding the job specification method and modifying the user interface. This process is the best way to develop and debug the functional part of your code, and it may be all you need to do. Modifying GridSample is explained in detail in "Overriding Job Specification" (page 27).

To integrate Xgrid capability into an existing application or to create an Xgrid client from scratch, use the Cocoa API for Xgrid: Xgrid Foundation. The process is described in "Writing a Cocoa Xgrid Client" (page 35). Even if this is your intended goal, you will probably save time and effort by using the `xgrid` command-line client and modifying the GridSample code as part of the development process, before creating or modifying your own application using Xgrid Foundation.

Using the Xgrid Command-Line Client

Mac OS X version 10.4 and later includes the `xgrid` command-line utility. For some applications, the `xgrid` command-line tool is all the client software you need. In any event, you should become familiar with it to get a better understanding of Xgrid before writing an Xgrid-enabled client application.

The `xgrid` command-line utility is an Xgrid client. You can submit a job to a controller by typing xgrid followed by the controller's host name and a job specification.

The job specification includes the name and path of an executable file, such as an application or a shell script, and any arguments to pass to the executable file.

You have the option of supplying an input file or a directory of files. If you supply an input directory, it is copied to each agent and becomes the working directory for the executable file.

You also have the option of specifying an output file or directory.

Important: You have the option of providing a relative path or an absolute path when specifying executable files, input files and directories, and output files and directories. When a relative path is used, the executable and the input files or directories are copied to the agents, and the output files or directories are created for every agent and collected by the controller. If you specify an *absolute* path to the executable, input, or output files or directories, those files are assumed to exist on the agent computers, or to be available to the agents as part of a shared file system, at the path location specified. They are not copied or created.

As each agent completes its task, the standard output and error streams are returned to the controller. You can pipe these streams to files or direct them to the output and error streams of the shell that submitted the job. You can also retrieve any other files created in the agent's working directory during job execution. These files are returned to the output directory specified when the job is submitted.

Basic xgrid Syntax

If you type `man xgrid` from within the Terminal application, you see an illustration of the syntax for the command-line tool. The first few lines are shown in Listing 3-1 (page 17).

Listing 3-1 Excerpt from the xgrid man page

```
SYNOPSIS
       xgrid [-h[ostname] hostname] [-auth { Password | Kerberos }] [-p[assword]
 password]
        xgrid -job run [-gid grid-identifier] [-si stdin] [-in indir] [-so
stdout] [-se stderr] [-out outdir]
        [-email email-address]
       cmd [arg1 [...]]
       xgrid -job submit [-gid grid-identifier] [-si stdin] [-in indir] [-dids
 jobid [, jobid]*]
```

```
        [-email email-address]
        cmd [arg1 [...]]
```

The first parameter is `-h`, followed by the host name or IP address of the controller.

Example: `xgrid -h localhost`

You can optionally include a method of authentication and a password.

The next parameter of interest is the job specification. You can either run a job synchronously, by passing `-job run`, or submit a job for asynchronous execution by passing `-job submit`. In either case, you must then specify a grid, and can optionally redirect the standard input and output.

If you submit a job asynchronously, rather than running it synchronously, you can include an email address to be notified when the job terminates.

Running a Job Synchronously

Here's a very simple example that runs the `cal` program using a controller on the local host:

Listing 3-2 Running a job synchronously

```
xgrid -h localhost -job run /usr/bin/cal 2007
```

By specifying the full path, you prevent the executable file from being copied to the agent. Instead, it is run in place at the specified path location. No working directory is created.

By specifying `run` instead of `submit`, you tell `xgrid` to execute the command synchronously. The command line returns nothing until the job is complete.

Since the optional input and output specifications have been omitted, standard output is used for the results.

Submitting a Job for Asynchronous Execution

Now let's look at a more complex example. It submits the file `myscript` with a group of files in an input directory. An email address is passed that will be used to notify someone at every job state change. The results are saved in files in an output directory, then the job is deleted:

Listing 3-3 Submitting a job asynchronously

```
$ xgrid -job submit -in ~/data/working -email somebody@apple.com myscript param1
 param2
          { jobIdentifier = 27; }
          $ xgrid -job results -id 27 -so job.out -se job.err -out job-outdir
           $ xgrid -job delete -id 27
```

In this example, `xgrid` is told to execute the job asynchronously by passing `submit` instead of `run`.

Use the `-in` parameter to pass an input directory. This directory is copied to each agent and becomes the working directory on the agent's host computer. You can include anything needed in the working directory, such as additonal input files, libraries, and executables. The executable file is run in this directory.

Next, specify the script to run, along with any input parameters. This completes the first line of input.

A job identifier is returned.

The next line of the example uses the job identifier to assign file names for the standard output, standard error, and job output for this particular job.

The last line deletes the job.

Submitting a Batch Job

Here's how to submit a batch job, consisting of multiple tasks. If enough agents are available, all of the tasks are performed simultaneously. Otherwise, each available agent is assigned a task and the first agent to finish is assigned another task, and so on until all the tasks have been completed.

To submit a batch job, you must include a property list file, describing each task to be performed. The `man` page for `xgrid` describes the structure of the property list file, but here's a helpful shortcut—submit each individual task as its own job initially, and let `xgrid` generate the property list file for you. When the task completes, you can retrieve the property list file. You can edit the list to modify a task, duplicating it as necessary, or concatenate the property list files of several tasks into one batch.

For example, if you submit the `cal` program as a job, it looks like this:

```
xgrid -job submit /usr/bin/cal  6 2007
```

The `xgrid` command returns a job ID. When the job completes, you can use the job identifier to retrieve the complete job specification, including the property list:

```
xgrid -job specification -id n
{
    jobSpecification = {
        applicationIdentifier = "com.apple.xgrid.cli";
        inputFiles = {};
        name = "/usr/bin/cal";
        submissionIdentifier = abc;
        taskSpecifications = {
            0 = {arguments = (6, 2007); command = "/usr/bin/cal"; };
       };
   };
}
```

Copy the returned job specification and save it using the `.plist` file suffix. You can then submit the file to Xgrid as part of a batch job specification. If the example above were named `batch.plist`, you could submit the job like this:

```
xgrid -job batch batch.plist
```

Before going any further with Xgrid programming, try breaking up your job into executable tasks and submitting them using the `xgrid` command-line utility. The experience will provide a great deal of useful information to you about how to plan and execute your program.

Building and Running GridSample

GridSample is not just a sample application. It is a library-quality set of code intended to handle a wide variety of job submission tasks in a robust fashion, with all the authentication, submission, notification, error handling, and housekeeping tasks handled correctly.

Consequently, it is not so much a simple teaching tool as an application framework. You can create your own application just by overwriting the job submission function and providing a nib file to customize the user interface.

The best way to learn from GridSample is not to study all of the code—there is a lot of it, and much of it is complex housekeeping—but rather to build and run it, then modify it as needed, studying the parts you need to modify.

This chapter covers the basics of building and using GridSample. Modifying the job specification function is discussed separately in "Overriding the Job Specification Function" (page 27).

The GridSample Targets

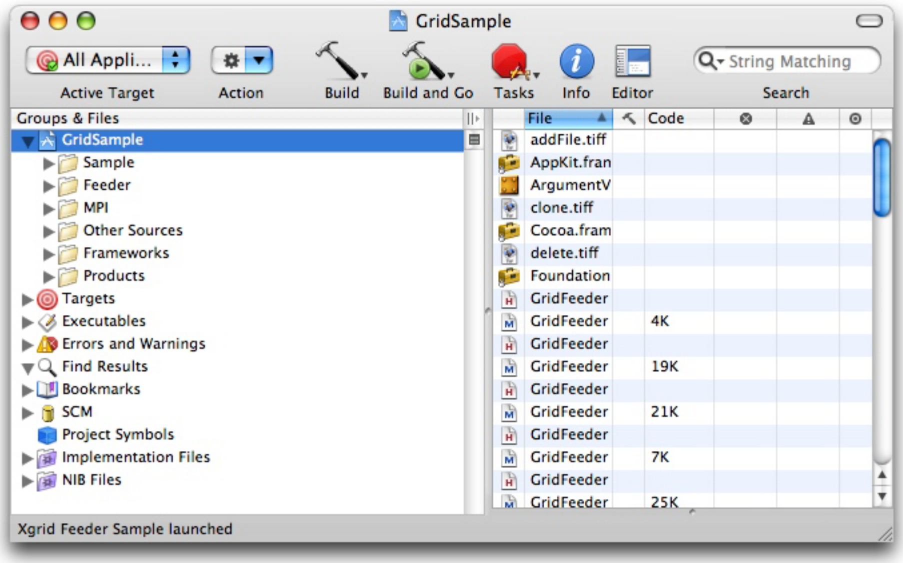
When you install the Mac OS X Developer Tools or Xcode Tools, a `Developer/Examples/Xgrid` directory is created. This directory contains the GridSample project. To get the current version, install the Developer Tools from the installation disc for Mac OS X 10.4 (Tiger) or the Xcode Tools for Mac OS X 10.5 (Leopard).

`GridSample.xcodeproj` is an Xcode project. Double-clicking it opens the project in Xcode.

Note: If the project does not open, you may have an older version of Project Builder installed. Install the latest version of the Mac OS X Developer Tools or Xcode Tools to get a current version of Xcode.

You should see the GridSample project, show in Figure 4-1.

Figure 4-1 GridSample project



Xgrid Feeder Sample launched

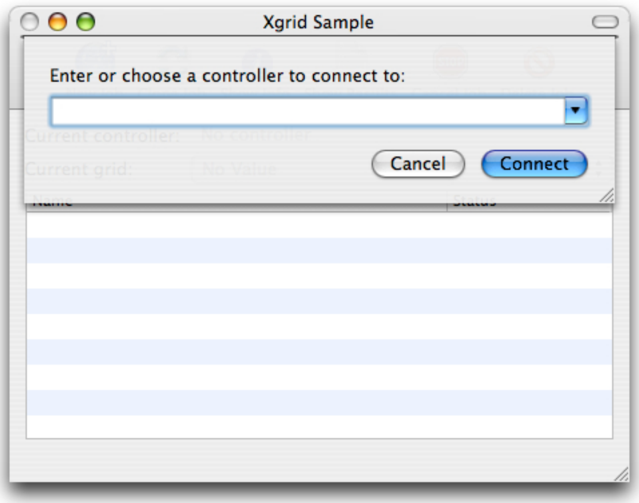
If you click the disclosure triangle next to the Targets icon, you will see that the project has three targets, Sample, Feeder, and MPI, which compile three complete applications: Xgrid Sample, Xgrid Feeder Sample, and Xgrid MPI Sample.

Xgrid Sample is the simplest application, and the easiest to use and understand. Xgrid Feeder Sample contains more sophisticated code for handling various types of job submission. Xgrid MPI Sample is intended to feed Xgrid jobs using MPI, a multiprocessing language which falls outside the scope of this document.

The Xgrid Sample Target

To build and run the Xgrid Sample application, select Xgrid Sample from the target list, then click the Build and Go icon in the toolbar at the top of the window. The code will compile, link, and run. You should see Figure 4-2, with a dialog prompting you to choose a controller.

Figure 4-2 The Xgrid Sample controller selection window

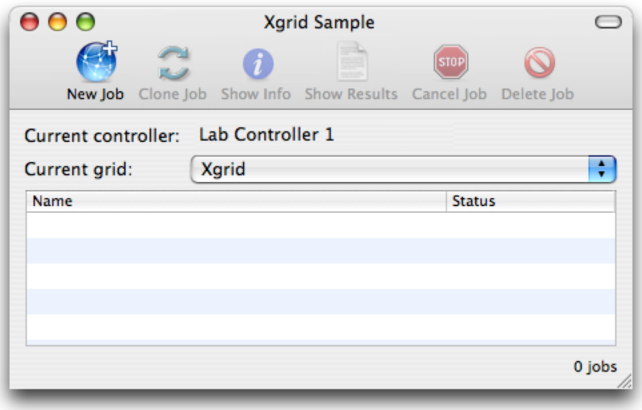


You must select a valid Xgrid controller in order to proceed.

Note: To run XgridSample, you should have access to a computer running Mac OS X Server as an Xgrid controller, as well as one or more additional computers acting as agents. For development purposes, however, it is possible to use a single computer running Mac OS X (one with 4 processors or cores recommended). To do this, download and install a copy of XgridLite to configure the controller daemon, enable the development system to act as an agent, and download and install the Mac OS X Server Admin tools on your development system. If you do this, you can run the sample code using your development system as client, controller, and agents, and be able to monitor and administer the process. The XgridLite application is shareware, however, and is not provided, supported, or endorsed by Apple.

Choose a controller from the pop-up list and enter the password. GridSample connects to the controller and brings up the user interface.

Figure 4-3 Xgrid Sample main window



Click the New Job icon to submit a job to Xgrid. You are then prompted for a job name and a command. The job name can be any descriptive string to help you identify the job. The syntax for the command is similar to a job submission for the `xgrid` command-line tool (for details, see "Using the xgrid Command-Line Client" (page 17)).

XgridSample submits the command to Xgrid as a single task, in the following format:

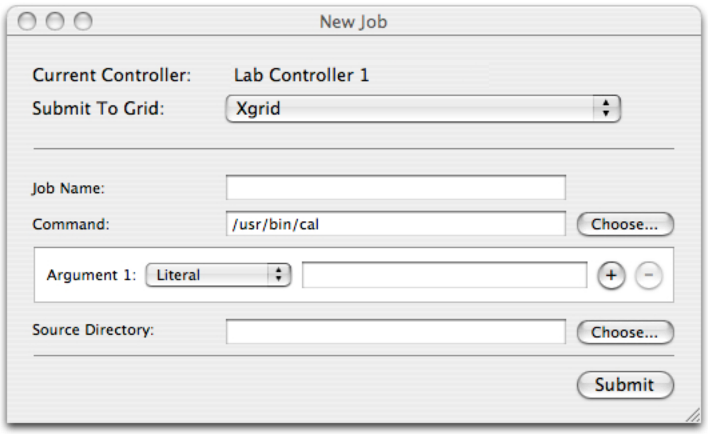
```
/bin/sh -c "COMMAND"
```

The GridFeeder Target

To build and run the Xgrid Feeder application, select Xgrid Feeder from the target list, then click the Build and Go icon in the toolbar at the top of the window. The code compiles, links, and runs. Again, you should see a window with a dialog prompting you to choose a controller.

Connect to a controller and the user interface screen appears. It is identical to the Grid Sample screen shown earlier. When you click the New Job icon, however, a far more sophisticated interface appears, allowing you to browse for executable tasks, construct a table of arguments, and choose a source directory for your job.

Figure 4-4 Gridfeeder Sample New Job window



Current Controller: Lab Controller 1
Submit To Grid: Xgrid
Job Name:
Command: /usr/bin/cal
Argument 1: Literal
Source Directory:

You can enter or choose a command, add arguments, browse for a source directory, and otherwise interactively construct a job entry for Xgrid.

Debugging and Monitoring Progress

Use the Xgrid Admin tool to debug your setup and monitor the job progress as Grid Sample or Grid Feeder executes your tasks.

You can download the Admin tools from http://www.apple.com/support/downloads/serveradmintools1047.html. Once you have downloaded and installed the Admin tools, Xgrid Admin is located in the Server subdirectory of your Applications directory.

Use Xgrid Admin to verify that you have a controller and agents available, and to monitor the status of the jobs you submit to Grid Sample or Grid Feeder.

Overriding the Job Specification Function

The easiest way to create an Xgrid client application is to override the job specification function in the GridSample sample code. This chapter shows you how to accomplish this.

How the GridSample Code Is Organized

The GridSample application is organized into several modules. All the modules are part of the GridSample project, found in your `Developer/Examples/Xgrid/GridSample/` directory.

Some modules consist entirely of `.nib` files that open in Interface Builder. Most of the modules are paired `.h` and `.m` files that open in the code editor—`GridSampleApplicationDelegate.h` and `GridSampleApplicationDelegat.m`, for example.

The GridSample modules form an outline of the main tasks any Xgrid client must perform:

- Identify a controller—You need to locate a controller, typically by opening a service browser window.

  See the `GridSampleConnectionController` and `GridSampleServiceBrowser` modules.
- Authenticate and connect—You need to connect to the controller, which is typically password protected.

  See the `GridSampleLoginController` module.
- Submit the job—You need to assemble the tasks into a job and submit the job to the controller, specifying a grid.

  See the `GridSampleNewJobWindowController` and `NewJob.nib` modules.
- Monitor status and retrieve job ID—When you submit a job, an action monitor object is returned. You need to check the status of the action monitor to see if your job was submitted successfully. If it was, you can obtain a job ID.

  See the `GridSampleNewJobWindowController` module.
- Set callback for notifications—Using the job ID, you can register to be notified when the job completes or when errors occur.

  See the `GridSampleNewJobWindowController` module.
- Collect data—When the job completes, you need to collect the returned data and deal with it appropriately.

  See the `GridSampleResultsWindowController` module.
- Housekeeping—As always, there are error handling routines and basic housekeeping tasks, such as deleting the job.

  See the `dealloc` function in the `GridSampleResultsWindowController` module.

The GridSample application is general purpose. In addition to modifying the job submission module, you may want to streamline other behaviors. For example:

- You may want to connect to a specific controller every time. To modify this behavior, refer to the GridSampleConnectionController and GridSampleServiceBrowser modules.
- You may want to use only your chosen method of authentication. To modify this behavior, refer to the GridSampleLoginController module.
- You may want to direct the collected data to another application for postprocessing. To modify this behavior, refer to the GridSampleJobResultsWindowController module.

Most of these modifications are fairly straightforward. Examining the source code of the appropriate module should provide you with much of the information you need, and referring the XgridFoundation Reference should answer remaining questions. The job submission module deserves special attention, however.

Changing the Job Specification

To override the job specification function in Grid Sample, you need to make modifications in four places:

- `NewJob.nib`
- `NewJobWindowController` (`.m` and `.h`)
- `ApplicationDelegate` (`.m` and `.h`)
- `MainMenu.nib`

For example, if you examine the GridCalendar sample code, found at http://developer.apple.com/samplecode/GridCalendar/, you will see that it is a copy of the GridSample application, with the following modifications:

- A new `NewJob.nib` file has been created, with a new UI for job specification.
- The `jobSpecification` function has been subclassed to create a window controller that builds a job specification based on the new UI.
- A new application delegate has been created, subclassing `classForNewJobWindowController` to point to the new window controller.
- The application delegate has been subclassed in `MainMenu.nib` to specify the new application delegate.

Here are the steps in more detail:

- `NewJob.nib` has been changed to provide a new job submission user interface. (The `.nib` file is created using Interface Builder without actually writing any code.)
- *GridSample*`NewJobWindowController` has been supplemented with *GridCalendar*`NewJobWindowController`, which contains the support code to process the UI state returned from `NewJob.nib` into a properly formatted job submission. The new support code is encapsulated as a subclass of `jobSpecification`, overriding the job specification code in `GridSampleNewJobWindowController`.

- *GridSample*`ApplicationDelegate` has been supplemented with *GridCalendar*`ApplicationDelegate`, which overrides `-classForNewJobWindowController` to point to `GridCalendarNewJobWindowController`.
- In `Mainmenu.nib`, the application delegate object's class has been subclassed to `GridCalendarApplicationDelegate`.

You should proceed in a similar manner, using the code in GridCalendar as a guide. The process is broken down into four steps:

Note: If you are new to Cocoa programming, refer to *Cocoa Application Tutorial* for guidance.

Step One: Create a New Job Submission UI

Create a new job submission UI by modifying and saving `NewJob.nib`.

Note that the GridSample project contains three `NewJob.nib` files, one for each target: GridSample, GridFeeder, and GridMPI. Open the .nib file GridSample > Sample > Resources > Nibs, as shown in Figure 5-1.

Figure 5-1 Locating the right `NewJob.nib` file

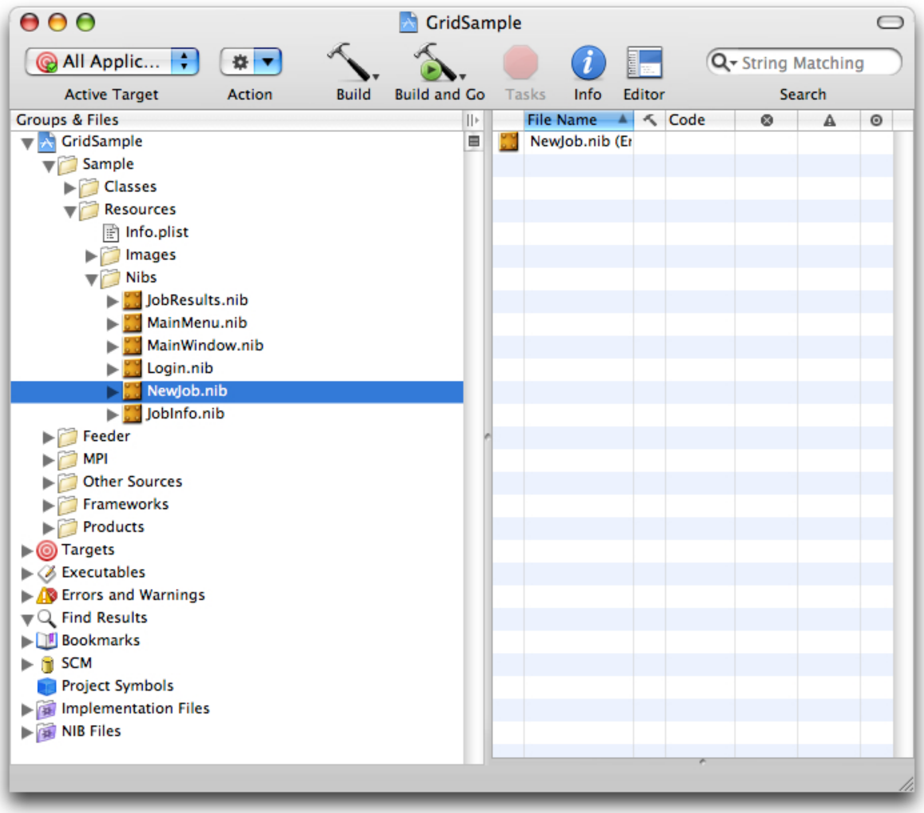


Figure 5-2 and Figure 5-3 (page 30)show the NewJob.nib file from GridSample and the modified NewJob.nib file for GridCalendar. Since GridCalendar always uses the same command, the command field and job name field have been removed, and a date selector field has been added for the `cal` command.

Figure 5-2 GridSample job submission `.nib` file

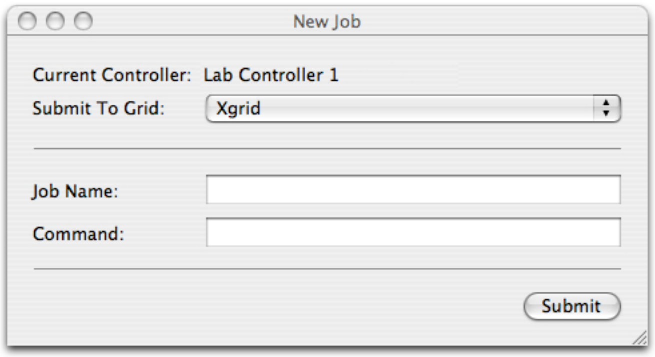
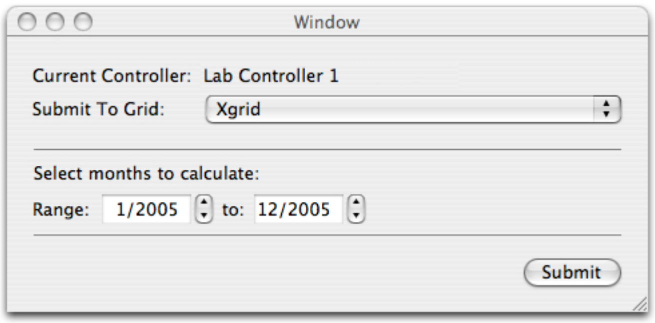


Figure 5-3 GridCalendar job submission `.nib` file


Window
Current Controller: Lab Controller 1
Submit To Grid: Xgrid
Select months to calculate:
Range: 1/2005 to: 12/2005
Submit

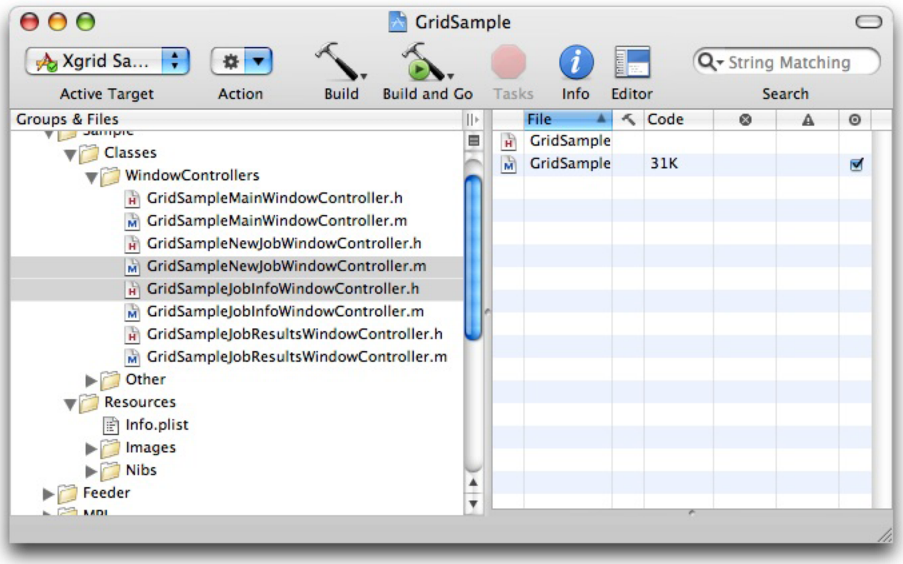
Assuming your application always submits the same type of job to Xgrid, you should make similar changes to the user interface: keep the pop-up window to select the grid, but replace the job name and command fields with input fields that allow the user to set the parameters for each instance of your particular job.

Note: If you are unfamiliar with building `.nib` files using Interface Builder, refer to *Cocoa Application Tutorial* for guidance.

Step Two: Add the Support Code

Create a new file that contains the support logic to create a job specification based on the returned state from your UI. Override `jobSpecification`, as defined in `GridSampleNewJobWindowController.m`, by creating a new class of the same name, encapsulating your code. This file supplements `GridSampleNewJobWindowController`, so name it appropriately (in GridCalendar, it's named `GridCalendarNewJobWindowController`).

Figure 5-4 GridSampleNewJobWindowContoller files


GridSample
Xgrid Sa...
Active Target
Action
Build
Build and Go
Tasks
Info
Editor
String Matching
Search
Groups & Files
Classes
WindowControllers
GridSampleMainWindowController.h
GridSampleMainWindowController.m
GridSampleNewJobWindowController.h
GridSampleNewJobWindowController.m
GridSampleJobInfoWindowController.h
GridSampleJobInfoWindowController.m
GridSampleJobResultsWindowController.h
GridSampleJobResultsWindowController.m
Other
Resources
Info.plist
Images
Nibs
Feeder
File
Code
GridSample
GridSample 31K

To see how this is done, compare `GridSampleNewJobWindowController.m` and with `GridCalendarNewJobWindowController.m`. Notice that `GridCalendarNewJobWindowController.m` contains only the modified job specification code, and that the main function is named `jobSpecification`, overriding the `jobSpecification` defined in `GridSampleNewJobWindowController.m`. This way, the new window controller inherits all the functionality of the old window controller, such as selecting a grid, submitting the job, retrieving the action monitor, and setting up callbacks for status. Only the job specification function is overridden.

Step Three: Create an Application Delegate

Supplement `GridSampleApplicationDelegate.m` with your own application delegate that sublasses `classForNewJobWindowController` to point to your new window controller, thereby overriding `GridSampleNewJobWindowController` with your own window controller.

Listing 5-1 shows the contents of the `GridCalendarApplicationDelegate.m` file in its entirety.

Listing 5-1 GridCalendarApplicationDelegate

```
#import "GridCalendarApplicationDelegate.h"
#import "GridCalendarNewJobWindowController.h"
@implementation GridCalendarApplicationDelegate
#pragma mark *** Accessor methods ***
- (Class)classForNewJobWindowController;
{
    return [GridCalendarNewJobWindowController self];

}
@end
```

As you can see, the code is quite simple. As the implementation of your application delegate, define `classForNewJobWindowController` to return an instance of the window controller you added in step 2.

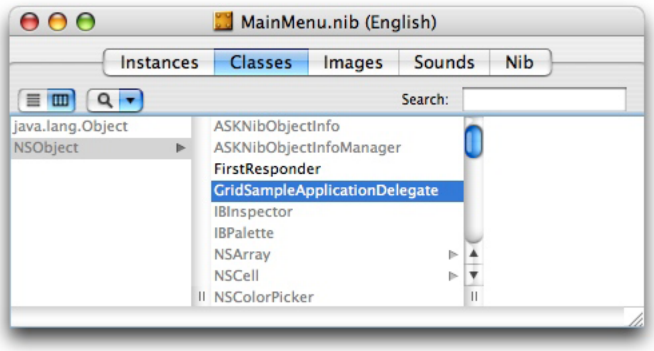
Step Four: Change the Application Delegate's Class in MainMenu.nib

Finally, change the application delegate's class in `MainMenu.nib` to point to the new application delegate you added in step 3.

1. Open `MainMenu.nib` (There are three `.nib` files by this name in GridSample—one for each target. Open the `MainMenu.nib` file for GridSample.)

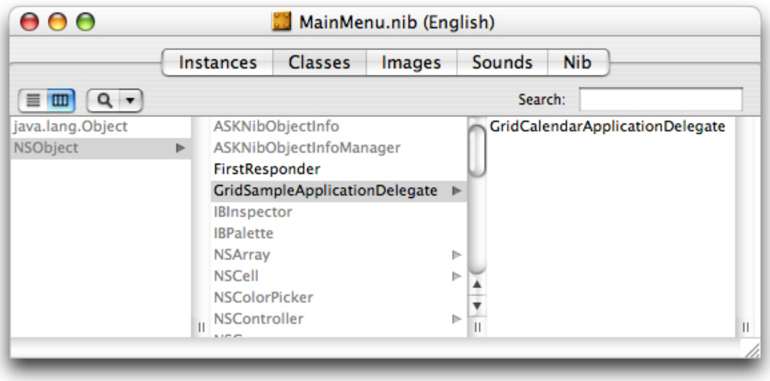
2. Select the Classes tab and highlight `GridSampleApplicationDelegate`, as shown in Figure 5-5.

Figure 5-5 Classes tab in `MainMenu.nib`



3. Press the Return key. This creates a new subclass, `MyGridSampleApplicationDelegate`. Double-click the new entry to select it, then type the name of your application delegate from step 3. You will see your new application delegate subclass, similar to Figure 5-6.

Figure 5-6 `GridCalendarApplicationDelegate` subclass



4. Save your changes, then click the Build and Go icon to compile and run your application. You should have a customized Xgrid client application that submits your job to Xgrid.

Summary

To summarize: you have created a `.nib` file with a new UI for job specification; you have subclassed `jobSpecification` to create a window controller that builds a job specification based on your UI; you have created an application delegate and subclassed `classForNewJobWindowController` to point to your window controller; and you have subclassed the application delegate in the `MainMenu.nib` file to use your new application delegate.

Congratulations. Unless you need to further customize the look and feel of your application, you're done. If you do need to go further, see "Writing a Cocoa Xgrid Client" (page 35).

Writing a Cocoa Xgrid Client

As you learned in the previous chapter, you can create a client application by simply modifying the GridSample sample code. You may need to go further, however, and develop your own application, or add Xgrid capability to an existing application. This chapter describes the basic steps.

Writing an Application in Six Steps

There are six steps that any Xgrid client application must take:

1. Locate a controller—You need to locate a controller, which typically involves bringing up a service browser and letting the user choose a controller.
2. Connect and authenticate—You need to connect to the controller, which is typically password protected.
3. Submit the job—The tasks must be assembled and submitted to the controller with the proper syntax, specifying a grid.
4. Identify the job—When you submit a job, an action monitor object is returned. Use this object to determine whether your job submission was successful and to obtain the job ID for future reference.
5. Receive notifications—Once you have a job ID, you can register to be notified when the job completes, when tasks complete, or when errors occur.
6. Collect the data—When the job completes, you need to collect the returned data and deal with it appropriately.

Of course, there are housekeeping tasks as well, such as deleting the job when you are done.

This chapter highlights the functions you need to use to accomplish each step, and points you to code samples that illustrate their use.

All code samples are included as part of the GridSample project, which is installed in the `Developer/Examples/Xcode` directory on your hard disk when you install the Mac OS X Developer Tools (Tiger) or XCode Tools (Leopard).

Important: Most objects in the Xgrid Foundation API load their attributes asynchronously. If you obtain an object from a function, and immediately interrogate its attributes, you may find that they are mostly `nil`. To retrieve valid attributes, set up a callback using `XGActionMonitor`; when you receive a notification, test the `isUpdating` attribute. You may get multiple notifications before updating is complete. Keep monitoring until the `isUpdating` attribute is false. The other attributes are now valid and ready for you to work with.

Step One: Locate a Controller and Connect

To browse for Xgrid controllers, use the `NSNetServiceBrowser` function. `GridSampleServiceBrowser.m` in GridSample provides a working code sample of an Xgrid service browser.

Connect to an Xgrid server using the `XGConnection` and `XGController` functions. See `GridSampleConnectionController.m` in GridSample for working code samples.

Step Two: Authenticate

You may not need to authenticate to access an Xgrid controller, but you typically do.

Authentication by means of a simple password is supported, as is authentication using single sign-on (SSO or Kerberos). The `XGConnection` function requires an authentication parameter. If the controller is unprotected, pass `nil`. If it is protected, pass the user name and password, using `XGTwoWayRandomAuthenticator`. If using Kerberos, use `XGGSSAuthenticator`.

`GridSampleConnectionController.m` attempts an initial connection with the connection authenticator set to `nil`. If the connection fails with an authentication error, the user is prompted for login information and `XGAuthenticator` is set.

Refer to `GridSampleLoginController.m` in GridSample for a working code sample.

Step Three: Submit a Job

Submit jobs to Xgrid using the `XGController` function, using this syntax:

```
-[XGController performSubmitJobActionWithJobSpecification:gridIdentifier:]
```

Note that you need to pass a job specification and a grid identifier, of type `XGGrid`.

`GridSampleNewJobWindow.m` includes sample code for a pop-up window that allows the user to choose an available grid.

Constructing a properly formatted job submission is non-trivial. Refer to the man page for `xgrid` for the correct syntax.

Refer to `GridSampleNewJobWindowController.m` in GridSample for working job submission sample code. You may also find `GridCalendarNewJobWindowController.m` helpful. If you have not already done so, download the GridCalendar sample code from http://developer.apple.com/samplecode/GridCalendar/.

Step Four: Retrieve Job ID

When you submit a job using `XGController`, an `XGActionMonitor` object is returned. Monitor the `XGActionMonitor` object until the `isUpdating` attribute is false, then verify that the dictionary contains a "success" entry. If the job submission is successful, the `jobIdentifier` attribute contains the job ID, an object of type `XGJob`. Pass this identifier to the appropriate functions to monitor the job status and retrieve the data.

Refer to `GridSampleNewJobWindowController.m` in GridSample for working sample code.

Step Five: Register for Notifications

Register for notification of job state changes, or query the controller for the current job state, using the `XGActionMonitor` function. Pass in the job identifier you received in step four.

Refer to `GridSampleNewJobWindowController.m` in GridSample for working sample code.

Step Six: Data Collection

When the job completes, collect your data using `XGFile` and `XGFileDownload` functions. Identify the job using the identifier object (`XGJob`).

Refer to `GridSampleJobResultsWindowController.m` in GridSample for working sample code.

Good Housekeeping

When the job is complete, and you have collected the results, the job should be deleted, notification observers should be removed, and unneeded structures and objects should be deallocated.

See the `dealloc` function in `GridSampleJobResultsWindowController.m` for an example of good housekeeping practices.

Document Revision History

This table describes the changes to *Xgrid Programming Guide*.

Date	Notes
2007-10-31	New document describes the programming interfaces to Xgrid, Apple's technology for distributed multiprocessing.