```
/*
Copyright 2011 Google Inc.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
nYou may obtain a copy of the License at

     http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/

// Package magic implements MIME type sniffing of data based on the
// well-known "magic" number prefixes in the file.
package magic

import (
      "bytes"
      "io"
      "net/http"
      "strings"
)

type prefixEntry struct {
      prefix []byte
      mtype  string
}

// usable source: http://www.garykessler.net/library/file_sigs.html
// mime types: http://www.iana.org/assignments/media-types/media-types.xhtml
var prefixTable = []prefixEntry{
      {[]byte("GIF87a"), "image/gif"},
      {[]byte("GIF89a"), "image/gif"}, // TODO: Others?
      {[]byte("\xff\xd8\xff\xe2"), "image/jpeg"},
      {[]byte("\xff\xd8\xff\xe1"), "image/jpeg"},
      {[]byte("\xff\xd8\xff\xe0"), "image/jpeg"},
      {[]byte("\xff\xd8\xff\xdb"), "image/jpeg"},
      {[]byte("\x49\x49\x2a\x00\x10\x00\x00\x00\x43\x52\x02"), "image/cr2"},
      {[]byte{137, 'P', 'N', 'G', '\r', '\n', 26, 10}, "image/png"},
      {[]byte{0x49, 0x20, 0x49}, "image/tiff"},
      {[]byte{0x49, 0x49, 0x2A, 0}, "image/tiff"},
      {[]byte{0x4D, 0x4D, 0, 0x2A}, "image/tiff"},
      {[]byte{0x4D, 0x4D, 0, 0x2B}, "image/tiff"},
      {[]byte("8BPS"), "image/vnd.adobe.photoshop"},
      {[]byte("gimp xcf "), "image/xcf"},
      {[]byte("-----BEGIN PGP PUBLIC KEY BLOCK---"), "text/x-openpgp-public-
key"},
      {[]byte("fLaC\x00\x00\x00"), "audio/flac"},
      {[]byte{'I', 'D', '3'}, "audio/mpeg"},
      {[]byte{0, 0, 1, 0xB7}, "video/mpeg"},
      {[]byte{0, 0, 0, 0x14, 0x66, 0x74, 0x79, 0x70, 0x71, 0x74, 0x20, 0x20},
"video/quicktime"},
      {[]byte{0, 0x6E, 0x1E, 0xF0}, "application/vnd.ms-powerpoint"},
      {[]byte{0x1A, 0x45, 0xDF, 0xA3}, "video/webm"},
      {[]byte("FLV\x01"), "application/vnd.adobe.flash.video"},
      {[]byte{0x1F, 0x8B, 0x08}, "application/gzip"},
      {[]byte{0x37, 0x7A, 0xBC, 0xAF, 0x27, 0x1C}, "application/x-7z-
compressed"},
```

```
	{[]byte("BZh"), "application/bzip2"},
	{[]byte{0xFD, 0x37, 0x7A, 0x58, 0x5A, 0}, "application/x-xz"},
	{[]byte{'P', 'K', 3, 4, 0x0A, 0, 2, 0}, "application/epub+zip"},
	{[]byte{0xD0, 0xCF, 0x11, 0xE0, 0xA1, 0xB1, 0x1A, 0xE1},
"application/vnd.ms-word"},
	{[]byte{'P', 'K', 3, 4, 0x0A, 0x14, 0, 6, 0},
"application/vnd.openxmlformats-officedocument.custom-properties+xml"},
	{[]byte{'P', 'K', 3, 4}, "application/zip"},
	{[]byte("%PDF"), "application/pdf"},
	{[]byte("{rtf"), "text/rtf1"},
	{[]byte("BEGIN:VCARD\x0D\x0A"), "text/vcard"},
	{[]byte("Return-Path: "), "message/rfc822"},

	// TODO(bradfitz): popular audio & video formats at least
}

// MIMEType returns the MIME type from the data in the provided header
// of the data.
// It returns the empty string if the MIME type can't be determined.
func MIMEType(hdr []byte) string {
	hlen := len(hdr)
	for _, pte := range prefixTable {
		plen := len(pte.prefix)
		if hlen > plen && bytes.Equal(hdr[:plen], pte.prefix) {
			return pte.mtype
		}
	}
	t := http.DetectContentType(hdr)
	t = strings.Replace(t, "; charset=utf-8", "", 1)
	if t != "application/octet-stream" && t != "text/plain" {
		return t
	}
	return ""
}

// MIMETypeFromReader takes a reader, sniffs the beginning of it,
// and returns the mime (if sniffed, else "") and a new reader
// that's the concatenation of the bytes sniffed and the remaining
// reader.
func MIMETypeFromReader(r io.Reader) (mime string, reader io.Reader) {
	var buf bytes.Buffer
	io.CopyN(&buf, r, 1024)
	mime = MIMEType(buf.Bytes())
	return mime, io.MultiReader(&buf, r)
}

// MIMETypeFromReader takes a ReaderAt, sniffs the beginning of it,
// and returns the MIME type if sniffed, else the empty string.
func MIMETypeFromReaderAt(ra io.ReaderAt) (mime string) {
	var buf [1024]byte
	n, _ := ra.ReadAt(buf[:], 0)
	return MIMEType(buf[:n])
}
```