```
static boolean ReadICCProfile(j_decompress_ptr jpeg_info)
{
  char
    magick[12];

  ErrorManager
    *error_manager;

  ExceptionInfo
    *exception;

  Image
    *image;

  MagickBooleanType
    status;

  register ssize_t
    i;

  register unsigned char
    *p;

  size_t
    length;

  StringInfo
    *icc_profile,
    *profile;

  /*
    Read color profile.
  */
  length=(size_t) ((size_t) GetCharacter(jpeg_info) << 8);
  length+=(size_t) GetCharacter(jpeg_info);
  length-=2;
  if (length <= 14)
    {
      while (length-- > 0)
        if (GetCharacter(jpeg_info) == EOF)
          break;
      return(TRUE);
    }
  for (i=0; i < 12; i++)
    magick[i]=(char) GetCharacter(jpeg_info);
  if (LocaleCompare(magick,ICC_PROFILE) != 0)
    {
      /*
        Not a ICC profile, return.
      */
      for (i=0; i < (ssize_t) (length-12); i++)
        if (GetCharacter(jpeg_info) == EOF)
          break;
      return(TRUE);
```

```
    }
  (void) GetCharacter(jpeg_info);  /* id */
  (void) GetCharacter(jpeg_info);  /* markers */
  length-=14;
  error_manager=(ErrorManager *) jpeg_info->client_data;
  exception=error_manager->exception;
  image=error_manager->image;
  profile=BlobToStringInfo((const void *) NULL,length);
  if (profile == (StringInfo *) NULL)
    {
      (void) ThrowMagickException(exception,GetMagickModule(),
        ResourceLimitError,"MemoryAllocationFailed","`%s'",image-
>filename);
      return(FALSE);
    }
  error_manager->profile=profile;
  p=GetStringInfoDatum(profile);
  for (i=0; i < (ssize_t) length; i++)
  {
    int
      c;

    c=GetCharacter(jpeg_info);
    if (c == EOF)
      break;
    *p++=(unsigned char) c;
  }
  if (i != (ssize_t) length)
    {
      profile=DestroyStringInfo(profile);
      (void) ThrowMagickException(exception,GetMagickModule(),
        CorruptImageError,"InsufficientImageDataInFile","`%s'",
        image->filename);
      return(FALSE);
    }
  error_manager->profile=NULL;
  icc_profile=(StringInfo *) GetImageProfile(image,"icc");
  if (icc_profile != (StringInfo *) NULL)
    {
      ConcatenateStringInfo(icc_profile,profile);
      profile=DestroyStringInfo(profile);
    }
  else
    {
      status=SetImageProfile(image,"icc",profile,exception);
      profile=DestroyStringInfo(profile);
      if (status == MagickFalse)
        {
          (void) ThrowMagickException(exception,GetMagickModule(),
            ResourceLimitError,"MemoryAllocationFailed","`%s'",image-
>filename);
          return(FALSE);
        }
    }
```

```
  if (image->debug != MagickFalse)
    (void) LogMagickEvent(CoderEvent,GetMagickModule(),
      "Profile: ICC, %.20g bytes",(double) length);
  return(TRUE);
}
<sep>
zzip_mem_disk_load(ZZIP_MEM_DISK* dir, ZZIP_DISK* disk)
{
    if (dir->list) zzip_mem_disk_unload(dir);
    ___ struct zzip_disk_entry* entry = zzip_disk_findfirst(disk);
    for (; entry ; entry = zzip_disk_findnext(disk, entry)) {
      ZZIP_MEM_DISK_ENTRY* item = zzip_mem_disk_entry_new(disk, entry);
      if (dir->last) { dir->last->zz_next = item; }
      else { dir->list = item; }; dir->last = item;
    } ____;
    dir->disk = disk;
    return 0;
}
<sep>
int rsa_pkcs1_decrypt( rsa_context *ctx,
                      int mode, size_t *olen,
                      const unsigned char *input,
                      unsigned char *output,
                      size_t output_max_len)
{
    switch( ctx->padding )
    {
        case RSA_PKCS_V15:
            return rsa_rsaes_pkcs1_v15_decrypt( ctx, mode, olen, input,
output,
                                                output_max_len );

#if defined(POLARSSL_PKCS1_V21)
        case RSA_PKCS_V21:
            return rsa_rsaes_oaep_decrypt( ctx, mode, NULL, 0, olen,
input,
                                           output, output_max_len );
#endif

        default:
            return( POLARSSL_ERR_RSA_INVALID_PADDING );
    }
}
<sep>
static int ehci_process_itd(EHCIState *ehci,
                            EHCIitd *itd,
                            uint32_t addr)
{
    USBDevice *dev;
    USBEndpoint *ep;
    uint32_t i, len, pid, dir, devaddr, endp;
    uint32_t pg, off, ptr1, ptr2, max, mult;

    ehci->periodic_sched_active = PERIODIC_ACTIVE;
```

```
dir =(itd->bufptr[1] & ITD_BUFPTR_DIRECTION);
devaddr = get_field(itd->bufptr[0], ITD_BUFPTR_DEVADDR);
endp = get_field(itd->bufptr[0], ITD_BUFPTR_EP);
max = get_field(itd->bufptr[1], ITD_BUFPTR_MAXPKT);
mult = get_field(itd->bufptr[2], ITD_BUFPTR_MULT);

for(i = 0; i < 8; i++) {
    if (itd->transact[i] & ITD_XACT_ACTIVE) {
        pg   = get_field(itd->transact[i], ITD_XACT_PGSEL);
        off  = itd->transact[i] & ITD_XACT_OFFSET_MASK;
        len  = get_field(itd->transact[i], ITD_XACT_LENGTH);

        if (len > max * mult) {
            len = max * mult;
        }
        if (len > BUFF_SIZE || pg > 6) {
            return -1;
        }

        ptr1 = (itd->bufptr[pg] & ITD_BUFPTR_MASK);
        qemu_sglist_init(&ehci->isgl, ehci->device, 2, ehci->as);
        if (off + len > 4096) {
            /* transfer crosses page border */
            if (pg == 6) {
                qemu_sglist_destroy(&ehci->isgl);
                return -1;  /* avoid page pg + 1 */
            }
            ptr2 = (itd->bufptr[pg + 1] & ITD_BUFPTR_MASK);
            uint32_t len2 = off + len - 4096;
            uint32_t len1 = len - len2;
            qemu_sglist_add(&ehci->isgl, ptr1 + off, len1);
            qemu_sglist_add(&ehci->isgl, ptr2, len2);
        } else {
            qemu_sglist_add(&ehci->isgl, ptr1 + off, len);
        }

        dev = ehci_find_device(ehci, devaddr);
        if (dev == NULL) {
            ehci_trace_guest_bug(ehci, "no device found");
            qemu_sglist_destroy(&ehci->isgl);
            return -1;
        }
        pid = dir ? USB_TOKEN_IN : USB_TOKEN_OUT;
        ep = usb_ep_get(dev, pid, endp);
        if (ep && ep->type == USB_ENDPOINT_XFER_ISOC) {
            usb_packet_setup(&ehci->ipacket, pid, ep, 0, addr, false,
                             (itd->transact[i] & ITD_XACT_IOC) != 0);
            usb_packet_map(&ehci->ipacket, &ehci->isgl);
            usb_handle_packet(dev, &ehci->ipacket);
            usb_packet_unmap(&ehci->ipacket, &ehci->isgl);
        } else {
            DPRINTF("ISOCH: attempt to addess non-iso endpoint\n");
            ehci->ipacket.status = USB_RET_NAK;
```

```
                ehci->ipacket.actual_length = 0;
            }
            qemu_sglist_destroy(&ehci->isgl);

            switch (ehci->ipacket.status) {
            case USB_RET_SUCCESS:
                break;
            default:
                fprintf(stderr, "Unexpected iso usb result: %d\n",
                        ehci->ipacket.status);
                /* Fall through */
            case USB_RET_IOERROR:
            case USB_RET_NODEV:
                /* 3.3.2: XACTERR is only allowed on IN transactions */
                if (dir) {
                    itd->transact[i] |= ITD_XACT_XACTERR;
                    ehci_raise_irq(ehci, USBSTS_ERRINT);
                }
                break;
            case USB_RET_BABBLE:
                itd->transact[i] |= ITD_XACT_BABBLE;
                ehci_raise_irq(ehci, USBSTS_ERRINT);
                break;
            case USB_RET_NAK:
                /* no data for us, so do a zero-length transfer */
                ehci->ipacket.actual_length = 0;
                break;
            }
            if (!dir) {
                set_field(&itd->transact[i], len - ehci-
>ipacket.actual_length,
                          ITD_XACT_LENGTH); /* OUT */
            } else {
                set_field(&itd->transact[i], ehci->ipacket.actual_length,
                          ITD_XACT_LENGTH); /* IN */
            }
            if (itd->transact[i] & ITD_XACT_IOC) {
                ehci_raise_irq(ehci, USBSTS_INT);
            }
            itd->transact[i] &= ~ITD_XACT_ACTIVE;
        }
    }
    return 0;
}
<sep>
GF_Filter *gf_fs_load_filter(GF_FilterSession *fsess, const char *name,
GF_Err *err_code)
{
      const char *args=NULL;
      const char *sep, *file_ext;
      u32 i, len, count = gf_list_count(fsess->registry);
      Bool quiet = (err_code && (*err_code == GF_EOS)) ? GF_TRUE :
GF_FALSE;
```

```
	assert(fsess);
	assert(name);
	if (err_code) *err_code = GF_OK;

	sep = gf_fs_path_escape_colon(fsess, name);
	if (sep) {
		args = sep+1;
		len = (u32) (sep - name);
	} else len = (u32) strlen(name);

	if (!len) {
		if (!quiet) {
			GF_LOG(GF_LOG_ERROR, GF_LOG_FILTER, ("Missing filter
name in %s\n", name));
		}
		return NULL;
	}

	if (!strncmp(name, "enc", len)) {
		return gf_fs_load_encoder(fsess, args);
	}
	/*regular filter loading*/
	for (i=0;i<count;i++) {
		const GF_FilterRegister *f_reg = gf_list_get(fsess->registry,
i);
		if ((strlen(f_reg->name)==len) && !strncmp(f_reg->name, name,
len)) {
			GF_Filter *filter;
			GF_FilterArgType argtype = GF_FILTER_ARG_EXPLICIT;

			if ((f_reg->flags & GF_FS_REG_REQUIRES_RESOLVER) &&
!fsess->max_resolve_chain_len) {
				GF_LOG(GF_LOG_ERROR, GF_LOG_FILTER, ("Filter %s
requires graph resolver but it is disabled\n", name));
				if (err_code) *err_code = GF_BAD_PARAM;
				return NULL;
			}

			if (f_reg->flags & GF_FS_REG_ACT_AS_SOURCE) argtype =
GF_FILTER_ARG_EXPLICIT_SOURCE;
			filter = gf_filter_new(fsess, f_reg, args, NULL,
argtype, err_code, NULL, GF_FALSE);
			if (!filter) return NULL;
			if (!filter->num_output_pids) {
				const char *src_url = strstr(name, "src");
				if (src_url && (src_url[3]==fsess->sep_name))
					gf_filter_post_process_task(filter);
			}
			return filter;
		}
	}
	/*check JS file*/
	file_ext = gf_file_ext_start(name);
	if (file_ext && (file_ext > sep) )
```

```
            file_ext = NULL;

      if (!file_ext || strstr(name, ".js") || strstr(name, ".jsf") ||
strstr(name, ".mjs") ) {
            Bool file_exists = GF_FALSE;
            char szName[10+GF_MAX_PATH];
            char szPath[10+GF_MAX_PATH];
            if (len>GF_MAX_PATH)
                  return NULL;

            strncpy(szPath, name, len);
            szPath[len]=0;
            GF_LOG(GF_LOG_DEBUG, GF_LOG_FILTER, ("Trying JS filter %s\n",
szPath));
            if (gf_file_exists(szPath)) {
                  file_exists = GF_TRUE;
            } else {
                  strcpy(szName, szPath);
                  file_exists = gf_fs_solve_js_script(szPath, szName,
file_ext);
                  if (!file_exists && !file_ext) {
                        strcat(szName, ".js");
                        if (gf_file_exists(szName)) {
                              strncpy(szPath, name, len);
                              szPath[len]=0;
                              strcat(szPath, ".js");
                              file_exists = GF_TRUE;
                        }
                  }
            }

            if (file_exists) {
                  sprintf(szName, "jsf%cjs%c", fsess->sep_args, fsess-
>sep_name);
                  strcat(szName, szPath);
                  if (name[len])
                        strcat(szName, name+len);
                  return gf_fs_load_filter(fsess, szName, err_code);
            }
      }

      if (!quiet) {
            GF_LOG(GF_LOG_ERROR, GF_LOG_FILTER, ("Failed to load filter
%s: no such filter registry\n", name));
      }
      if (err_code) *err_code = GF_FILTER_NOT_FOUND;
      return NULL;
}
<sep>
  void Compute(OpKernelContext* context) override {
    // Read ragged_splits inputs.
    OpInputList ragged_nested_splits_in;
    OP_REQUIRES_OK(context, context->input_list("rt_nested_splits",
```

```
&ragged_nested_splits_in));
    const int ragged_nested_splits_len = ragged_nested_splits_in.size();
    RaggedTensorVariant batched_ragged_input;
    // Read ragged_values input.
    batched_ragged_input.set_values(context-
>input(ragged_nested_splits_len));
    batched_ragged_input.mutable_nested_splits()->reserve(
        ragged_nested_splits_len);
    for (int i = 0; i < ragged_nested_splits_len; i++) {
      batched_ragged_input.append_splits(ragged_nested_splits_in[i]);
    }

    if (!batched_input_) {
      // Encode as a Scalar Variant Tensor.
      Tensor* encoded_scalar;
      OP_REQUIRES_OK(context, context->allocate_output(0,
TensorShape({}),
                                                       &encoded_scalar));
      encoded_scalar->scalar<Variant>()() =
std::move(batched_ragged_input);
      return;
    }

    // Unbatch the Ragged Tensor and encode the components.
    std::vector<RaggedTensorVariant> unbatched_ragged_input;
    auto batched_splits_top_vec =
        batched_ragged_input.splits(0).vec<SPLIT_TYPE>();
    int num_components = batched_splits_top_vec.size() - 1;
    OP_REQUIRES(context, num_components >= 0,
                errors::Internal("Invalid split argument."));
    OP_REQUIRES_OK(context, UnbatchRaggedZerothDim<VALUE_TYPE,
SPLIT_TYPE>(
                                batched_ragged_input,
&unbatched_ragged_input));

    // Bundle the encoded scalar Variant Tensors into a rank-1 Variant
Tensor.
    Tensor* encoded_vector;
    int output_size = unbatched_ragged_input.size();
    OP_REQUIRES_OK(context,
                   context->allocate_output(0,
TensorShape({output_size}),
                                            &encoded_vector));
    auto encoded_vector_t = encoded_vector->vec<Variant>();
    for (int i = 0; i < output_size; i++) {
      encoded_vector_t(i) = unbatched_ragged_input[i];
    }
  }
<sep>
static bool check_underflow(const struct ip6t_entry *e)
{
      const struct xt_entry_target *t;
      unsigned int verdict;
```

```
      if (!unconditional(&e->ipv6))
            return false;
      t = ip6t_get_target_c(e);
      if (strcmp(t->u.user.name, XT_STANDARD_TARGET) != 0)
            return false;
      verdict = ((struct xt_standard_target *)t)->verdict;
      verdict = -verdict - 1;
      return verdict == NF_DROP || verdict == NF_ACCEPT;
}
<sep>
SegmentCommand* Binary::segment_from_offset(uint64_t offset) {
  return const_cast<SegmentCommand*>(static_cast<const Binary*>(this)-
>segment_from_offset(offset));
}
<sep>
create_backup (char const *to, const struct stat *to_st, bool
leave_original)
{
  /* When the input to patch modifies the same file more than once, patch
only
     backs up the initial version of each file.

     To figure out which files have already been backed up, patch
remembers the
     files that replace the original files.  Files not known already are
backed
     up; files already known have already been backed up before, and are
     skipped.

     When a patch tries to delete a file, in order to not break the above
     logic, we merely remember which file to delete.  After the entire
patch
     file has been read, we delete all files marked for deletion which
have not
     been recreated in the meantime.  */

  if (to_st && ! (S_ISREG (to_st->st_mode) || S_ISLNK (to_st->st_mode)))
    fatal ("File %s is not a %s -- refusing to create backup",
        to, S_ISLNK (to_st->st_mode) ? "symbolic link" : "regular
file");

  if (to_st && lookup_file_id (to_st) == CREATED)
    {
      if (debug & 4)
      say ("File %s already seen\n", quotearg (to));
    }
  else
    {
      int try_makedirs_errno = 0;
      char *bakname;

      if (origprae || origbase || origsuff)
      {
```

```
        char const *p = origprae ? origprae : "";
        char const *b = origbase ? origbase : "";
        char const *s = origsuff ? origsuff : "";
        char const *t = to;
        size_t plen = strlen (p);
        size_t blen = strlen (b);
        size_t slen = strlen (s);
        size_t tlen = strlen (t);
        char const *o;
        size_t olen;

        for (o = t + tlen, olen = 0;
             o > t && ! ISSLASH (*(o - 1));
             o--)
          /* do nothing */ ;
        olen = t + tlen - o;
        tlen -= olen;
        bakname = xmalloc (plen + tlen + blen + olen + slen + 1);
        memcpy (bakname, p, plen);
        memcpy (bakname + plen, t, tlen);
        memcpy (bakname + plen + tlen, b, blen);
        memcpy (bakname + plen + tlen + blen, o, olen);
        memcpy (bakname + plen + tlen + blen + olen, s, slen + 1);

        if ((origprae
             && (contains_slash (origprae + FILE_SYSTEM_PREFIX_LEN
(origprae))
                 || contains_slash (to)))
            || (origbase && contains_slash (origbase)))
          try_makedirs_errno = ENOENT;
      }
      else
      {
        bakname = find_backup_file_name (to, backup_type);
        if (!bakname)
          xalloc_die ();
      }

      if (! to_st)
      {
        int fd;

        if (debug & 4)
          say ("Creating empty file %s\n", quotearg (bakname));

        try_makedirs_errno = ENOENT;
        safe_unlink (bakname);
        while ((fd = safe_open (bakname, O_CREAT | O_WRONLY | O_TRUNC,
0666)) < 0)
          {
            if (errno != try_makedirs_errno)
              pfatal ("Can't create file %s", quotearg (bakname));
            makedirs (bakname);
            try_makedirs_errno = 0;
```

```
          }
        if (close (fd) != 0)
          pfatal ("Can't close file %s", quotearg (bakname));
      }
      else if (leave_original)
      create_backup_copy (to, bakname, to_st, try_makedirs_errno == 0);
      else
      {
        if (debug & 4)
          say ("Renaming file %s to %s\n",
             quotearg_n (0, to), quotearg_n (1, bakname));
        while (safe_rename (to, bakname) != 0)
          {
            if (errno == try_makedirs_errno)
            {
              makedirs (bakname);
              try_makedirs_errno = 0;
            }
            else if (errno == EXDEV)
            {
              create_backup_copy (to, bakname, to_st,
                                  try_makedirs_errno == 0);
              safe_unlink (to);
              break;
            }
            else
            pfatal ("Can't rename file %s to %s",
                  quotearg_n (0, to), quotearg_n (1, bakname));
          }
      }
      free (bakname);
    }
}
<sep>
int iscsi_session_get_param(struct iscsi_cls_session *cls_session,
                      enum iscsi_param param, char *buf)
{
      struct iscsi_session *session = cls_session->dd_data;
      int len;

      switch(param) {
      case ISCSI_PARAM_FAST_ABORT:
            len = sprintf(buf, "%d\n", session->fast_abort);
            break;
      case ISCSI_PARAM_ABORT_TMO:
            len = sprintf(buf, "%d\n", session->abort_timeout);
            break;
      case ISCSI_PARAM_LU_RESET_TMO:
            len = sprintf(buf, "%d\n", session->lu_reset_timeout);
            break;
      case ISCSI_PARAM_TGT_RESET_TMO:
            len = sprintf(buf, "%d\n", session->tgt_reset_timeout);
            break;
      case ISCSI_PARAM_INITIAL_R2T_EN:
```

```
		len = sprintf(buf, "%d\n", session->initial_r2t_en);
		break;
	case ISCSI_PARAM_MAX_R2T:
		len = sprintf(buf, "%hu\n", session->max_r2t);
		break;
	case ISCSI_PARAM_IMM_DATA_EN:
		len = sprintf(buf, "%d\n", session->imm_data_en);
		break;
	case ISCSI_PARAM_FIRST_BURST:
		len = sprintf(buf, "%u\n", session->first_burst);
		break;
	case ISCSI_PARAM_MAX_BURST:
		len = sprintf(buf, "%u\n", session->max_burst);
		break;
	case ISCSI_PARAM_PDU_INORDER_EN:
		len = sprintf(buf, "%d\n", session->pdu_inorder_en);
		break;
	case ISCSI_PARAM_DATASEQ_INORDER_EN:
		len = sprintf(buf, "%d\n", session->dataseq_inorder_en);
		break;
	case ISCSI_PARAM_DEF_TASKMGMT_TMO:
		len = sprintf(buf, "%d\n", session->def_taskmgmt_tmo);
		break;
	case ISCSI_PARAM_ERL:
		len = sprintf(buf, "%d\n", session->erl);
		break;
	case ISCSI_PARAM_TARGET_NAME:
		len = sprintf(buf, "%s\n", session->targetname);
		break;
	case ISCSI_PARAM_TARGET_ALIAS:
		len = sprintf(buf, "%s\n", session->targetalias);
		break;
	case ISCSI_PARAM_TPGT:
		len = sprintf(buf, "%d\n", session->tpgt);
		break;
	case ISCSI_PARAM_USERNAME:
		len = sprintf(buf, "%s\n", session->username);
		break;
	case ISCSI_PARAM_USERNAME_IN:
		len = sprintf(buf, "%s\n", session->username_in);
		break;
	case ISCSI_PARAM_PASSWORD:
		len = sprintf(buf, "%s\n", session->password);
		break;
	case ISCSI_PARAM_PASSWORD_IN:
		len = sprintf(buf, "%s\n", session->password_in);
		break;
	case ISCSI_PARAM_IFACE_NAME:
		len = sprintf(buf, "%s\n", session->ifacename);
		break;
	case ISCSI_PARAM_INITIATOR_NAME:
		len = sprintf(buf, "%s\n", session->initiatorname);
		break;
	case ISCSI_PARAM_BOOT_ROOT:
```

```
		len = sprintf(buf, "%s\n", session->boot_root);
		break;
	case ISCSI_PARAM_BOOT_NIC:
		len = sprintf(buf, "%s\n", session->boot_nic);
		break;
	case ISCSI_PARAM_BOOT_TARGET:
		len = sprintf(buf, "%s\n", session->boot_target);
		break;
	case ISCSI_PARAM_AUTO_SND_TGT_DISABLE:
		len = sprintf(buf, "%u\n", session->auto_snd_tgt_disable);
		break;
	case ISCSI_PARAM_DISCOVERY_SESS:
		len = sprintf(buf, "%u\n", session->discovery_sess);
		break;
	case ISCSI_PARAM_PORTAL_TYPE:
		len = sprintf(buf, "%s\n", session->portal_type);
		break;
	case ISCSI_PARAM_CHAP_AUTH_EN:
		len = sprintf(buf, "%u\n", session->chap_auth_en);
		break;
	case ISCSI_PARAM_DISCOVERY_LOGOUT_EN:
		len = sprintf(buf, "%u\n", session->discovery_logout_en);
		break;
	case ISCSI_PARAM_BIDI_CHAP_EN:
		len = sprintf(buf, "%u\n", session->bidi_chap_en);
		break;
	case ISCSI_PARAM_DISCOVERY_AUTH_OPTIONAL:
		len = sprintf(buf, "%u\n", session->discovery_auth_optional);
		break;
	case ISCSI_PARAM_DEF_TIME2WAIT:
		len = sprintf(buf, "%d\n", session->time2wait);
		break;
	case ISCSI_PARAM_DEF_TIME2RETAIN:
		len = sprintf(buf, "%d\n", session->time2retain);
		break;
	case ISCSI_PARAM_TSID:
		len = sprintf(buf, "%u\n", session->tsid);
		break;
	case ISCSI_PARAM_ISID:
		len = sprintf(buf, "%02x%02x%02x%02x%02x%02x\n",
			      session->isid[0], session->isid[1],
			      session->isid[2], session->isid[3],
			      session->isid[4], session->isid[5]);
		break;
	case ISCSI_PARAM_DISCOVERY_PARENT_IDX:
		len = sprintf(buf, "%u\n", session->discovery_parent_idx);
		break;
	case ISCSI_PARAM_DISCOVERY_PARENT_TYPE:
		if (session->discovery_parent_type)
			len = sprintf(buf, "%s\n",
				      session->discovery_parent_type);
		else
			len = sprintf(buf, "\n");
		break;
```

```
      default:
            return -ENOSYS;
      }

      return len;
}
<sep>
gs_manager_create_windows_for_screen (GSManager *manager,
                                      GdkScreen *screen)
{
        GSWindow *window;
        int       n_monitors;
        int       i;

        g_return_if_fail (manager != NULL);
        g_return_if_fail (GS_IS_MANAGER (manager));
        g_return_if_fail (GDK_IS_SCREEN (screen));

        g_object_ref (manager);
        g_object_ref (screen);

        n_monitors = gdk_screen_get_n_monitors (screen);

        gs_debug ("Creating %d windows for screen %d", n_monitors,
gdk_screen_get_number (screen));

        for (i = 0; i < n_monitors; i++) {
                window = gs_window_new (screen, i, manager->priv-
>lock_active);

                gs_window_set_user_switch_enabled (window, manager->priv-
>user_switch_enabled);
                gs_window_set_logout_enabled (window, manager->priv-
>logout_enabled);
                gs_window_set_logout_timeout (window, manager->priv-
>logout_timeout);
                gs_window_set_logout_command (window, manager->priv-
>logout_command);
                gs_window_set_keyboard_enabled (window, manager->priv-
>keyboard_enabled);
                gs_window_set_keyboard_command (window, manager->priv-
>keyboard_command);
                gs_window_set_away_message (window, manager->priv-
>away_message);

                connect_window_signals (manager, window);

                manager->priv->windows = g_slist_append (manager->priv-
>windows, window);
        }

        g_object_unref (screen);
        g_object_unref (manager);
}
```

```
<sep>
static int complete_emulated_mmio(struct kvm_vcpu *vcpu)
{
      struct kvm_run *run = vcpu->run;
      struct kvm_mmio_fragment *frag;
      unsigned len;

      BUG_ON(!vcpu->mmio_needed);

      /* Complete previous fragment */
      frag = &vcpu->mmio_fragments[vcpu->mmio_cur_fragment];
      len = min(8u, frag->len);
      if (!vcpu->mmio_is_write)
            memcpy(frag->data, run->mmio.data, len);

      if (frag->len <= 8) {
            /* Switch to the next fragment. */
            frag++;
            vcpu->mmio_cur_fragment++;
      } else {
            /* Go forward to the next mmio piece. */
            frag->data += len;
            frag->gpa += len;
            frag->len -= len;
      }

      if (vcpu->mmio_cur_fragment == vcpu->mmio_nr_fragments) {
            vcpu->mmio_needed = 0;

            /* FIXME: return into emulator if single-stepping.  */
            if (vcpu->mmio_is_write)
                  return 1;
            vcpu->mmio_read_completed = 1;
            return complete_emulated_io(vcpu);
      }

      run->exit_reason = KVM_EXIT_MMIO;
      run->mmio.phys_addr = frag->gpa;
      if (vcpu->mmio_is_write)
            memcpy(run->mmio.data, frag->data, min(8u, frag->len));
      run->mmio.len = min(8u, frag->len);
      run->mmio.is_write = vcpu->mmio_is_write;
      vcpu->arch.complete_userspace_io = complete_emulated_mmio;
      return 0;
}
<sep>
static BOOL update_recv_secondary_order(rdpUpdate* update, wStream* s,
BYTE flags)
{
      BOOL rc = FALSE;
      size_t start, end, diff;
      BYTE orderType;
      UINT16 extraFlags;
      UINT16 orderLength;
```

```
	rdpContext* context = update->context;
	rdpSettings* settings = context->settings;
	rdpSecondaryUpdate* secondary = update->secondary;
	const char* name;

	if (Stream_GetRemainingLength(s) < 5)
	{
		WLog_Print(update->log, WLOG_ERROR,
"Stream_GetRemainingLength(s) < 5");
		return FALSE;
	}

	Stream_Read_UINT16(s, orderLength); /* orderLength (2 bytes) */
	Stream_Read_UINT16(s, extraFlags);  /* extraFlags (2 bytes) */
	Stream_Read_UINT8(s, orderType);    /* orderType (1 byte) */
	if (Stream_GetRemainingLength(s) < orderLength + 7U)
	{
		WLog_Print(update->log, WLOG_ERROR,
"Stream_GetRemainingLength(s) %" PRIuz " < %" PRIu16,
			Stream_GetRemainingLength(s), orderLength + 7);
		return FALSE;
	}

	start = Stream_GetPosition(s);
	name = secondary_order_string(orderType);
	WLog_Print(update->log, WLOG_DEBUG, "Secondary Drawing Order %s",
name);

	if (!check_secondary_order_supported(update->log, settings,
orderType, name))
		return FALSE;

	switch (orderType)
	{
		case ORDER_TYPE_BITMAP_UNCOMPRESSED:
		case ORDER_TYPE_CACHE_BITMAP_COMPRESSED:
		{
			const BOOL compressed = (orderType ==
ORDER_TYPE_CACHE_BITMAP_COMPRESSED);
			CACHE_BITMAP_ORDER* order =
			    update_read_cache_bitmap_order(update, s,
compressed, extraFlags);

			if (order)
			{
				rc = IFCALLRESULT(FALSE, secondary->CacheBitmap,
context, order);
				free_cache_bitmap_order(context, order);
			}
		}
		break;

		case ORDER_TYPE_BITMAP_UNCOMPRESSED_V2:
		case ORDER_TYPE_BITMAP_COMPRESSED_V2:
```

```
            {
                  const BOOL compressed = (orderType ==
ORDER_TYPE_BITMAP_COMPRESSED_V2);
                  CACHE_BITMAP_V2_ORDER* order =
                      update_read_cache_bitmap_v2_order(update, s,
compressed, extraFlags);

                  if (order)
                  {
                        rc = IFCALLRESULT(FALSE, secondary->CacheBitmapV2,
context, order);
                        free_cache_bitmap_v2_order(context, order);
                  }
            }
            break;

            case ORDER_TYPE_BITMAP_COMPRESSED_V3:
            {
                  CACHE_BITMAP_V3_ORDER* order =
update_read_cache_bitmap_v3_order(update, s, extraFlags);

                  if (order)
                  {
                        rc = IFCALLRESULT(FALSE, secondary->CacheBitmapV3,
context, order);
                        free_cache_bitmap_v3_order(context, order);
                  }
            }
            break;

            case ORDER_TYPE_CACHE_COLOR_TABLE:
            {
                  CACHE_COLOR_TABLE_ORDER* order =
                      update_read_cache_color_table_order(update, s,
extraFlags);

                  if (order)
                  {
                        rc = IFCALLRESULT(FALSE, secondary-
>CacheColorTable, context, order);
                        free_cache_color_table_order(context, order);
                  }
            }
            break;

            case ORDER_TYPE_CACHE_GLYPH:
            {
                  switch (settings->GlyphSupportLevel)
                  {
                        case GLYPH_SUPPORT_PARTIAL:
                        case GLYPH_SUPPORT_FULL:
                        {
                              CACHE_GLYPH_ORDER* order =
update_read_cache_glyph_order(update, s, extraFlags);
```

```
                              if (order)
                              {
                                    rc = IFCALLRESULT(FALSE, secondary-
>CacheGlyph, context, order);
                                    free_cache_glyph_order(context, order);
                              }
                        }
                        break;

                        case GLYPH_SUPPORT_ENCODE:
                        {
                              CACHE_GLYPH_V2_ORDER* order =
                                  update_read_cache_glyph_v2_order(update,
s, extraFlags);

                              if (order)
                              {
                                    rc = IFCALLRESULT(FALSE, secondary-
>CacheGlyphV2, context, order);
                                    free_cache_glyph_v2_order(context,
order);
                              }
                        }
                        break;

                        case GLYPH_SUPPORT_NONE:
                        default:
                              break;
                  }
            }
            break;

            case ORDER_TYPE_CACHE_BRUSH:
                  /* [MS-RDPEGDI] 2.2.2.2.1.2.7 Cache Brush
(CACHE_BRUSH_ORDER) */
                  {
                        CACHE_BRUSH_ORDER* order =
update_read_cache_brush_order(update, s, extraFlags);

                        if (order)
                        {
                              rc = IFCALLRESULT(FALSE, secondary-
>CacheBrush, context, order);
                              free_cache_brush_order(context, order);
                        }
                  }
                  break;

            default:
                  WLog_Print(update->log, WLOG_WARN, "SECONDARY ORDER %s
not supported", name);
                  break;
      }
```

```
      if (!rc)
      {
            WLog_Print(update->log, WLOG_ERROR, "SECONDARY ORDER %s
failed", name);
      }

      start += orderLength + 7;
      end = Stream_GetPosition(s);
      if (start > end)
      {
            WLog_Print(update->log, WLOG_WARN, "SECONDARY_ORDER %s: read
%" PRIuz "bytes too much",
                       name, end - start);
            return FALSE;
      }
      diff = start - end;
      if (diff > 0)
      {
            WLog_Print(update->log, WLOG_DEBUG,
                       "SECONDARY_ORDER %s: read %" PRIuz "bytes short,
skipping", name, diff);
            Stream_Seek(s, diff);
      }
      return rc;
}
<sep>
static int geneve_xmit_skb(struct sk_buff *skb, struct net_device *dev,
                     struct geneve_dev *geneve,
                     const struct ip_tunnel_info *info)
{
      bool xnet = !net_eq(geneve->net, dev_net(geneve->dev));
      struct geneve_sock *gs4 = rcu_dereference(geneve->sock4);
      const struct ip_tunnel_key *key = &info->key;
      struct rtable *rt;
      struct flowi4 fl4;
      __u8 tos, ttl;
      __be16 df = 0;
      __be16 sport;
      int err;

      rt = geneve_get_v4_rt(skb, dev, gs4, &fl4, info);
      if (IS_ERR(rt))
            return PTR_ERR(rt);

      err = skb_tunnel_check_pmtu(skb, &rt->dst,
                           GENEVE_IPV4_HLEN + info->options_len,
                           netif_is_any_bridge_port(dev));
      if (err < 0) {
            dst_release(&rt->dst);
            return err;
      } else if (err) {
            struct ip_tunnel_info *info;
```

```
            info = skb_tunnel_info(skb);
            if (info) {
                  info->key.u.ipv4.dst = fl4.saddr;
                  info->key.u.ipv4.src = fl4.daddr;
            }

            if (!pskb_may_pull(skb, ETH_HLEN)) {
                  dst_release(&rt->dst);
                  return -EINVAL;
            }

            skb->protocol = eth_type_trans(skb, geneve->dev);
            netif_rx(skb);
            dst_release(&rt->dst);
            return -EMSGSIZE;
      }

      sport = udp_flow_src_port(geneve->net, skb, 1, USHRT_MAX, true);
      if (geneve->cfg.collect_md) {
            tos = ip_tunnel_ecn_encap(key->tos, ip_hdr(skb), skb);
            ttl = key->ttl;

            df = key->tun_flags & TUNNEL_DONT_FRAGMENT ? htons(IP_DF) :
0;
      } else {
            tos = ip_tunnel_ecn_encap(fl4.flowi4_tos, ip_hdr(skb), skb);
            if (geneve->cfg.ttl_inherit)
                  ttl = ip_tunnel_get_ttl(ip_hdr(skb), skb);
            else
                  ttl = key->ttl;
            ttl = ttl ? : ip4_dst_hoplimit(&rt->dst);

            if (geneve->cfg.df == GENEVE_DF_SET) {
                  df = htons(IP_DF);
            } else if (geneve->cfg.df == GENEVE_DF_INHERIT) {
                  struct ethhdr *eth = eth_hdr(skb);

                  if (ntohs(eth->h_proto) == ETH_P_IPV6) {
                        df = htons(IP_DF);
                  } else if (ntohs(eth->h_proto) == ETH_P_IP) {
                        struct iphdr *iph = ip_hdr(skb);

                        if (iph->frag_off & htons(IP_DF))
                              df = htons(IP_DF);
                  }
            }
      }

      err = geneve_build_skb(&rt->dst, skb, info, xnet, sizeof(struct
iphdr));
      if (unlikely(err))
            return err;

      udp_tunnel_xmit_skb(rt, gs4->sock->sk, skb, fl4.saddr, fl4.daddr,
```

```
                    tos, ttl, df, sport, geneve->cfg.info.key.tp_dst,
                    !net_eq(geneve->net, dev_net(geneve->dev)),
                    !(info->key.tun_flags & TUNNEL_CSUM));
      return 0;
}
<sep>
static void agent_connect(UdscsConnection *conn)
{
    struct agent_data *agent_data;
    agent_data = g_new0(struct agent_data, 1);
    GError *err = NULL;
    gint pid;

    if (session_info) {
        pid = vdagent_connection_get_peer_pid(VDAGENT_CONNECTION(conn),
&err);
        if (err || pid <= 0) {
            static const char msg[] = "Could not get peer PID,
disconnecting new client";
            if (err) {
                syslog(LOG_ERR, "%s: %s", msg, err->message);
                g_error_free(err);
            } else {
                syslog(LOG_ERR, "%s", msg);
            }
            agent_data_destroy(agent_data);
            udscs_server_destroy_connection(server, conn);
            return;
        }

        agent_data->session = session_info_session_for_pid(session_info,
pid);
    }

    g_object_set_data_full(G_OBJECT(conn), "agent_data", agent_data,
                           (GDestroyNotify) agent_data_destroy);
    udscs_write(conn, VDAGENTD_VERSION, 0, 0,
                (uint8_t *)VERSION, strlen(VERSION) + 1);
    update_active_session_connection(conn);

    if (device_info) {
        forward_data_to_session_agent(VDAGENTD_GRAPHICS_DEVICE_INFO,
                                      (uint8_t *) device_info,
device_info_size);
    }
}
<sep>
TEST(CudnnRNNOpsTest, ForwardLstm_ShapeFn) {
  int seq_length = 2;
  int batch_size = 3;
  int num_units = 4;
  int num_layers = 5;
  int dir_count = 1;
  std::vector<int> input_shape = {seq_length, batch_size, num_units};
```

```
  std::vector<int> input_h_shape = {num_layers * dir_count, batch_size,
                                    num_units};
  std::vector<int> output_shape = {seq_length, batch_size,
                                   num_units * dir_count};
  auto shape_to_str = [](const std::vector<int>& v) {
    return strings::StrCat("[", absl::StrJoin(v, ","), "]");
  };
  string input_shapes_desc = strings::StrCat(
      shape_to_str(input_shape), ";", shape_to_str(input_h_shape), ";",
      shape_to_str(input_h_shape), ";", "[?]");
  string output_shapes_desc = "[d0_0,d0_1,d1_2];in1;in1;?";

  ShapeInferenceTestOp op("CudnnRNN");
  TF_ASSERT_OK(NodeDefBuilder("test", "CudnnRNN")
                   .Input({"input", 0, DT_FLOAT})
                   .Input({"input_h", 0, DT_FLOAT})
                   .Input({"input_c", 0, DT_FLOAT})
                   .Input({"params", 0, DT_FLOAT})
                   .Attr("rnn_mode", "lstm")
                   .Attr("input_mode", "auto_select")
                   .Attr("direction", "unidirectional")
                   .Finalize(&op.node_def));
  INFER_OK(op, input_shapes_desc, output_shapes_desc);
}
<sep>
bool samba_private_attr_name(const char *unix_ea_name)
{
      static const char * const prohibited_ea_names[] = {
            SAMBA_POSIX_INHERITANCE_EA_NAME,
            SAMBA_XATTR_DOS_ATTRIB,
            SAMBA_XATTR_MARKER,
            XATTR_NTACL_NAME,
            NULL
      };

      int i;

      for (i = 0; prohibited_ea_names[i]; i++) {
            if (strequal( prohibited_ea_names[i], unix_ea_name))
                  return true;
      }
      if (strncasecmp_m(unix_ea_name, SAMBA_XATTR_DOSSTREAM_PREFIX,
                  strlen(SAMBA_XATTR_DOSSTREAM_PREFIX)) == 0) {
            return true;
      }
      return false;
}
<sep>
int ssh_scp_leave_directory(ssh_scp scp)
{
    char buffer[] = "E\n";
    int rc;
    uint8_t code;
```

```
    if (scp == NULL) {
        return SSH_ERROR;
    }

    if (scp->state != SSH_SCP_WRITE_INITED) {
        ssh_set_error(scp->session, SSH_FATAL,
                      "ssh_scp_leave_directory called under invalid
state");
        return SSH_ERROR;
    }

    rc = ssh_channel_write(scp->channel, buffer, strlen(buffer));
    if (rc == SSH_ERROR) {
        scp->state = SSH_SCP_ERROR;
        return SSH_ERROR;
    }

    rc = ssh_channel_read(scp->channel, &code, 1, 0);
    if (rc <= 0) {
        ssh_set_error(scp->session, SSH_FATAL, "Error reading status
code: %s",
                      ssh_get_error(scp->session));
        scp->state = SSH_SCP_ERROR;
        return SSH_ERROR;
    }

    if (code != 0) {
        ssh_set_error(scp->session, SSH_FATAL, "scp status code %ud not
valid",
                      code);
        scp->state = SSH_SCP_ERROR;
        return SSH_ERROR;
    }

    return SSH_OK;
}
<sep>
TEST_F(ConnectionHandlerTest, ContinueOnListenerFilterTimeout) {
  InSequence s;

  TestListener* test_listener =
      addListener(1, true, false, "test_listener",
Network::Address::SocketType::Stream,
                  std::chrono::milliseconds(15000), true);
  Network::MockListener* listener = new Network::MockListener();
  Network::ListenerCallbacks* listener_callbacks;
  EXPECT_CALL(dispatcher_, createListener_(_, _, _))
      .WillOnce(
          Invoke([&](Network::Socket&, Network::ListenerCallbacks& cb,
bool) -> Network::Listener* {
            listener_callbacks = &cb;
            return listener;
          }));
  EXPECT_CALL(test_listener->socket_, localAddress());
```

```
  handler_->addListener(*test_listener);

  Network::MockListenerFilter* test_filter = new
Network::MockListenerFilter();
  EXPECT_CALL(factory_, createListenerFilterChain(_))
      .WillRepeatedly(Invoke([&](Network::ListenerFilterManager& manager)
-> bool {
        manager.addAcceptFilter(Network::ListenerFilterPtr{test_filter});
        return true;
      }));
  EXPECT_CALL(*test_filter, onAccept(_))
      .WillOnce(Invoke([&](Network::ListenerFilterCallbacks&) ->
Network::FilterStatus {
        return Network::FilterStatus::StopIteration;
      }));
  Network::MockConnectionSocket* accepted_socket = new
NiceMock<Network::MockConnectionSocket>();
  Network::IoSocketHandleImpl io_handle{42};
  EXPECT_CALL(*accepted_socket,
ioHandle()).WillRepeatedly(ReturnRef(io_handle));
  Event::MockTimer* timeout = new Event::MockTimer(&dispatcher_);
  EXPECT_CALL(*timeout, enableTimer(std::chrono::milliseconds(15000),
_));
  listener_callbacks-
>onAccept(Network::ConnectionSocketPtr{accepted_socket});
  Stats::Gauge& downstream_pre_cx_active =
      stats_store_.gauge("downstream_pre_cx_active",
Stats::Gauge::ImportMode::Accumulate);
  EXPECT_EQ(1UL, downstream_pre_cx_active.value());

  EXPECT_CALL(manager_, findFilterChain(_)).WillOnce(Return(nullptr));
  EXPECT_CALL(*timeout, disableTimer());
  timeout->invokeCallback();
  dispatcher_.clearDeferredDeleteList();
  EXPECT_EQ(0UL, downstream_pre_cx_active.value());
  EXPECT_EQ(1UL,
stats_store_.counter("downstream_pre_cx_timeout").value());

  // Make sure we continued to try create connection.
  EXPECT_EQ(1UL, stats_store_.counter("no_filter_chain_match").value());

  EXPECT_CALL(*listener, onDestroy());
}
<sep>
static inline void ConvertLuvToXYZ(const double L,const double u,const
double v,
  double *X,double *Y,double *Z)
{
  double
    gamma;

  assert(X != (double *) NULL);
  assert(Y != (double *) NULL);
  assert(Z != (double *) NULL);
```

```
  if (L > (CIEK*CIEEpsilon))
    *Y=(double) pow((L+16.0)/116.0,3.0);
  else
    *Y=L/CIEK;

gamma=PerceptibleReciprocal((((52.0*L/(u+13.0*L*(4.0*D65X/(D65X+15.0*D65Y
+
    3.0*D65Z))))-1.0)/3.0)-(-1.0/3.0));

*X=gamma*((*Y*((39.0*L/(v+13.0*L*(9.0*D65Y/(D65X+15.0*D65Y+3.0*D65Z))))-
5.0))+
    5.0*(*Y));
  *Z=(*X*(((52.0*L/(u+13.0*L*(4.0*D65X/(D65X+15.0*D65Y+3.0*D65Z))))-
1.0)/3.0))-
    5.0*(*Y);
}
<sep>
__zzip_parse_root_directory(int fd,
                            struct _disk_trailer *trailer,
                            struct zzip_dir_hdr **hdr_return,
                            zzip_plugin_io_t io)
{
    auto struct zzip_disk_entry dirent;
    struct zzip_dir_hdr *hdr;
    struct zzip_dir_hdr *hdr0;
    uint16_t *p_reclen = 0;
    zzip_off64_t entries;
    zzip_off64_t zz_offset;     /* offset from start of root directory */
    char *fd_map = 0;
    zzip_off64_t zz_fd_gap = 0;
    zzip_off64_t zz_entries = _disk_trailer_localentries(trailer);
    zzip_off64_t zz_rootsize = _disk_trailer_rootsize(trailer);
    zzip_off64_t zz_rootseek = _disk_trailer_rootseek(trailer);
    __correct_rootseek(zz_rootseek, zz_rootsize, trailer);

    if (zz_entries < 0 || zz_rootseek < 0 || zz_rootseek < 0)
        return ZZIP_CORRUPTED;

    hdr0 = (struct zzip_dir_hdr *) malloc(zz_rootsize);
    if (! hdr0)
        return ZZIP_DIRSIZE;
    hdr = hdr0;
    __debug_dir_hdr(hdr);

    if (USE_MMAP && io->fd.sys)
    {
        zz_fd_gap = zz_rootseek & (_zzip_getpagesize(io->fd.sys) - 1);
        HINT4(" fd_gap=%ld, mapseek=0x%lx, maplen=%ld", (long)
(zz_fd_gap),
              (long) (zz_rootseek - zz_fd_gap),
              (long) (zz_rootsize + zz_fd_gap));
        fd_map =
            _zzip_mmap(io->fd.sys, fd, zz_rootseek - zz_fd_gap,
                       zz_rootsize + zz_fd_gap);
```

```
        /* if mmap failed we will fallback to seek/read mode */
        if (fd_map == MAP_FAILED)
        {
            NOTE2("map failed: %s", strerror(errno));
            fd_map = 0;
        } else
        {
            HINT3("mapped *%p len=%li", fd_map,
                  (long) (zz_rootsize + zz_fd_gap));
        }
    }

    for (entries=0, zz_offset=0; ; entries++)
    {
        register struct zzip_disk_entry *d;
        uint16_t u_extras, u_comment, u_namlen;

#     ifndef ZZIP_ALLOW_MODULO_ENTRIES
        if (entries >= zz_entries) {
            if (zz_offset + 256 < zz_rootsize) {
                FAIL4("%li's entry is long before the end of directory -
enable modulo_entries? (O:%li R:%li)",
                      (long) entries, (long) (zz_offset), (long)
zz_rootsize);
            }
            break;
        }
#     endif

        if (fd_map)
        {
            d = (void*)(fd_map+zz_fd_gap+zz_offset); /*
fd_map+fd_gap==u_rootseek */
        } else
        {
            if (io->fd.seeks(fd, zz_rootseek + zz_offset, SEEK_SET) < 0)
                return ZZIP_DIR_SEEK;
            if (io->fd.read(fd, &dirent, sizeof(dirent)) <
__sizeof(dirent))
                return ZZIP_DIR_READ;
            d = &dirent;
        }

        if ((zzip_off64_t) (zz_offset + sizeof(*d)) > zz_rootsize ||
            (zzip_off64_t) (zz_offset + sizeof(*d)) < 0)
        {
            FAIL4("%li's entry stretches beyond root directory (O:%li
R:%li)",
                  (long) entries, (long) (zz_offset), (long)
zz_rootsize);
            break;
        }

        if (! zzip_disk_entry_check_magic(d)) {
```

```
#        ifndef ZZIP_ALLOW_MODULO_ENTRIES
            FAIL4("%li's entry has no disk_entry magic indicator (O:%li
R:%li)",
                  (long) entries, (long) (zz_offset), (long)
zz_rootsize);
#        endif
            break;
        }

#       if 0 && defined DEBUG
        zzip_debug_xbuf((unsigned char *) d, sizeof(*d) + 8);
#       endif

        u_extras = zzip_disk_entry_get_extras(d);
        u_comment = zzip_disk_entry_get_comment(d);
        u_namlen = zzip_disk_entry_get_namlen(d);
        HINT5("offset=0x%lx, size %ld, dirent *%p, hdr %p\n",
              (long) (zz_offset + zz_rootseek), (long) zz_rootsize, d,
hdr);

        /* writes over the read buffer, Since the structure where data is
           copied is smaller than the data in buffer this can be done.
           It is important that the order of setting the fields is
considered
           when filling the structure, so that some data is not trashed
in
           first structure read.
           at the end the whole copied list of structures  is copied into
           newly allocated buffer */
        hdr->d_crc32 = zzip_disk_entry_get_crc32(d);
        hdr->d_csize = zzip_disk_entry_get_csize(d);
        hdr->d_usize = zzip_disk_entry_get_usize(d);
        hdr->d_off = zzip_disk_entry_get_offset(d);
        hdr->d_compr = zzip_disk_entry_get_compr(d);
        if (hdr->d_compr > _255)
            hdr->d_compr = 255;

        if ((zzip_off64_t) (zz_offset + sizeof(*d) + u_namlen) >
zz_rootsize ||
            (zzip_off64_t) (zz_offset + sizeof(*d) + u_namlen) < 0)
        {
            FAIL4("%li's name stretches beyond root directory (O:%li
N:%li)",
                  (long) entries, (long) (zz_offset), (long) (u_namlen));
            break;
        }

        if (fd_map)
            {  memcpy(hdr->d_name, fd_map+zz_fd_gap +
zz_offset+sizeof(*d), u_namlen); }
        else
            { io->fd.read(fd, hdr->d_name, u_namlen); }
        hdr->d_name[u_namlen] = '\0';
        hdr->d_namlen = u_namlen;
```

```
        /* update offset by the total length of this entry -> next entry
*/
        zz_offset += sizeof(*d) + u_namlen + u_extras + u_comment;

        if (zz_offset > zz_rootsize)
        {
            FAIL3("%li's entry stretches beyond root directory (O:%li)",
                  (long) entries, (long) (zz_offset));
            entries ++;
            break;
        }

        HINT5("file %ld { compr=%d crc32=$%x offset=%d",
              (long) entries, hdr->d_compr, hdr->d_crc32, hdr->d_off);
        HINT5("csize=%d usize=%d namlen=%d extras=%d",
              hdr->d_csize, hdr->d_usize, u_namlen, u_extras);
        HINT5("comment=%d name='%s' %s <sizeof %d> } ",
              u_comment, hdr->d_name, "", (int) sizeof(*d));

        p_reclen = &hdr->d_reclen;

        {
            register char *p = (char *) hdr;
            register char *q = aligned4(p + sizeof(*hdr) + u_namlen + 1);
            *p_reclen = (uint16_t) (q - p);
            hdr = (struct zzip_dir_hdr *) q;
        }
    }                           /*for */

    if (USE_MMAP && fd_map)
    {
        HINT3("unmap *%p len=%li", fd_map, (long) (zz_rootsize +
zz_fd_gap));
        _zzip_munmap(io->fd.sys, fd_map, zz_rootsize + zz_fd_gap);
    }

    if (p_reclen)
    {
        *p_reclen = 0;          /* mark end of list */

        if (hdr_return)
            *hdr_return = hdr0;
    }                           /* else zero (sane) entries */
#  ifndef ZZIP_ALLOW_MODULO_ENTRIES
    return (entries != zz_entries ? ZZIP_CORRUPTED : 0);
#  else
    return ((entries & (unsigned)0xFFFF) != zz_entries ? ZZIP_CORRUPTED :
0);
#  endif
}
<sep>
char* dd_load_text_ext(const struct dump_dir *dd, const char *name,
unsigned flags)
```

```
{
//    if (!dd->locked)
//        error_msg_and_die("dump_dir is not opened"); /* bug */

    if (!str_is_correct_filename(name))
    {
        error_msg("Cannot load text. '%s' is not a valid file name",
name);
        if (!(flags & DD_LOAD_TEXT_RETURN_NULL_ON_FAILURE))
            xfunc_die();
    }

    /* Compat with old abrt dumps. Remove in abrt-2.1 */
    if (strcmp(name, "release") == 0)
        name = FILENAME_OS_RELEASE;

    char *full_path = concat_path_file(dd->dd_dirname, name);
    char *ret = load_text_file(full_path, flags);
    free(full_path);

    return ret;
}
<sep>
writeDataError(instanceData *pData, cJSON **pReplyRoot, uchar *reqmsg)
{
      char *rendered = NULL;
      cJSON *errRoot;
      cJSON *req;
      cJSON *replyRoot = *pReplyRoot;
      size_t toWrite;
      ssize_t wrRet;
      char errStr[1024];
      DEFiRet;

      if(pData->errorFile == NULL) {
            DBGPRINTF("omelasticsearch: no local error logger defined - "
                      "ignoring ES error information\n");
            FINALIZE;
      }

      if(pData->fdErrFile == -1) {
            pData->fdErrFile = open((char*)pData->errorFile,

      O_WRONLY|O_CREAT|O_APPEND|O_LARGEFILE|O_CLOEXEC,
                              S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP);
            if(pData->fdErrFile == -1) {
                  rs_strerror_r(errno, errStr, sizeof(errStr));
                  DBGPRINTF("omelasticsearch: error opening error file:
%s\n", errStr);
                  ABORT_FINALIZE(RS_RET_ERR);
            }
      }
      if((req=cJSON_CreateObject()) == NULL) ABORT_FINALIZE(RS_RET_ERR);
```

```
      cJSON_AddItemToObject(req, "url", cJSON_CreateString((char*)pData-
>restURL));
      cJSON_AddItemToObject(req, "postdata",
cJSON_CreateString((char*)reqmsg));

      if((errRoot=cJSON_CreateObject()) == NULL)
ABORT_FINALIZE(RS_RET_ERR);
      cJSON_AddItemToObject(errRoot, "request", req);
      cJSON_AddItemToObject(errRoot, "reply", replyRoot);
      rendered = cJSON_Print(errRoot);
      /* we do not do real error-handling on the err file, as this
finally complicates
       * things way to much.
       */
      DBGPRINTF("omelasticsearch: error record: '%s'\n", rendered);
      toWrite = strlen(rendered);
      wrRet = write(pData->fdErrFile, rendered, toWrite);
      if(wrRet != (ssize_t) toWrite) {
            DBGPRINTF("omelasticsearch: error %d writing error file,
write returns %lld\n",
                  errno, (long long) wrRet);
      }
      free(rendered);
      cJSON_Delete(errRoot);
      *pReplyRoot = NULL; /* tell caller not to delete once again! */

finalize_it:
      if(rendered != NULL)
            free(rendered);
      RETiRet;
}
<sep>
int sqlite3ExprCodeTarget(Parse *pParse, Expr *pExpr, int target){
  Vdbe *v = pParse->pVdbe;  /* The VM under construction */
  int op;                   /* The opcode being coded */
  int inReg = target;       /* Results stored in register inReg */
  int regFree1 = 0;         /* If non-zero free this temporary register
*/
  int regFree2 = 0;         /* If non-zero free this temporary register
*/
  int r1, r2;               /* Various register numbers */
  Expr tempX;               /* Temporary expression node */
  int p5 = 0;

  assert( target>0 && target<=pParse->nMem );
  if( v==0 ){
    assert( pParse->db->mallocFailed );
    return 0;
  }

expr_code_doover:
  if( pExpr==0 ){
    op = TK_NULL;
  }else{
```

```
    op = pExpr->op;
  }
  switch( op ){
    case TK_AGG_COLUMN: {
      AggInfo *pAggInfo = pExpr->pAggInfo;
      struct AggInfo_col *pCol = &pAggInfo->aCol[pExpr->iAgg];
      if( !pAggInfo->directMode ){
        assert( pCol->iMem>0 );
        return pCol->iMem;
      }else if( pAggInfo->useSortingIdx ){
        sqlite3VdbeAddOp3(v, OP_Column, pAggInfo->sortingIdxPTab,
                          pCol->iSorterColumn, target);
        return target;
      }
      /* Otherwise, fall thru into the TK_COLUMN case */
    }
    case TK_COLUMN: {
      int iTab = pExpr->iTable;
      if( ExprHasProperty(pExpr, EP_FixedCol) ){
        /* This COLUMN expression is really a constant due to WHERE
clause
        ** constraints, and that constant is coded by the pExpr->pLeft
        ** expresssion.  However, make sure the constant has the correct
        ** datatype by applying the Affinity of the table column to the
        ** constant.
        */
        int iReg = sqlite3ExprCodeTarget(pParse, pExpr->pLeft,target);
        int aff = sqlite3TableColumnAffinity(pExpr->y.pTab, pExpr-
>iColumn);
        if( aff>SQLITE_AFF_BLOB ){
          static const char zAff[] = "B\000C\000D\000E";
          assert( SQLITE_AFF_BLOB=='A' );
          assert( SQLITE_AFF_TEXT=='B' );
          if( iReg!=target ){
            sqlite3VdbeAddOp2(v, OP_SCopy, iReg, target);
            iReg = target;
          }
          sqlite3VdbeAddOp4(v, OP_Affinity, iReg, 1, 0,
                            &zAff[(aff-'B')*2], P4_STATIC);
        }
        return iReg;
      }
      if( iTab<0 ){
        if( pParse->iSelfTab<0 ){
          /* Other columns in the same row for CHECK constraints or
          ** generated columns or for inserting into partial index.
          ** The row is unpacked into registers beginning at
          ** 0-(pParse->iSelfTab).  The rowid (if any) is in a register
          ** immediately prior to the first column.
          */
          Column *pCol;
          Table *pTab = pExpr->y.pTab;
          int iSrc;
          int iCol = pExpr->iColumn;
```

```
          assert( pTab!=0 );
          assert( iCol>=XN_ROWID );
          assert( iCol<pExpr->y.pTab->nCol );
          if( iCol<0 ){
            return -1-pParse->iSelfTab;
          }
          pCol = pTab->aCol + iCol;
          testcase( iCol!=sqlite3TableColumnToStorage(pTab,iCol) );
          iSrc = sqlite3TableColumnToStorage(pTab, iCol) - pParse-
>iSelfTab;
#ifndef SQLITE_OMIT_GENERATED_COLUMNS
          if( pCol->colFlags & COLFLAG_GENERATED ){
            if( pCol->colFlags & COLFLAG_BUSY ){
              sqlite3ErrorMsg(pParse, "generated column loop on \"%s\"",
                              pCol->zName);
              return 0;
            }
            pCol->colFlags |= COLFLAG_BUSY;
            if( pCol->colFlags & COLFLAG_NOTAVAIL ){
              sqlite3ExprCodeGeneratedColumn(pParse, pCol, iSrc);
            }
            pCol->colFlags &= ~(COLFLAG_BUSY|COLFLAG_NOTAVAIL);
            return iSrc;
          }else
#endif /* SQLITE_OMIT_GENERATED_COLUMNS */
          if( pCol->affinity==SQLITE_AFF_REAL ){
            sqlite3VdbeAddOp2(v, OP_SCopy, iSrc, target);
            sqlite3VdbeAddOp1(v, OP_RealAffinity, target);
            return target;
          }else{
            return iSrc;
          }
        }else{
          /* Coding an expression that is part of an index where column
names
          ** in the index refer to the table to which the index belongs
*/
          iTab = pParse->iSelfTab - 1;
        }
      }
      return sqlite3ExprCodeGetColumn(pParse, pExpr->y.pTab,
                                pExpr->iColumn, iTab, target,
                                pExpr->op2);
    }
    case TK_INTEGER: {
      codeInteger(pParse, pExpr, 0, target);
      return target;
    }
    case TK_TRUEFALSE: {
      sqlite3VdbeAddOp2(v, OP_Integer, sqlite3ExprTruthValue(pExpr),
target);
      return target;
    }
#ifndef SQLITE_OMIT_FLOATING_POINT
```

```
    case TK_FLOAT: {
      assert( !ExprHasProperty(pExpr, EP_IntValue) );
      codeReal(v, pExpr->u.zToken, 0, target);
      return target;
    }
#endif
    case TK_STRING: {
      assert( !ExprHasProperty(pExpr, EP_IntValue) );
      sqlite3VdbeLoadString(v, target, pExpr->u.zToken);
      return target;
    }
    case TK_NULL: {
      sqlite3VdbeAddOp2(v, OP_Null, 0, target);
      return target;
    }
#ifndef SQLITE_OMIT_BLOB_LITERAL
    case TK_BLOB: {
      int n;
      const char *z;
      char *zBlob;
      assert( !ExprHasProperty(pExpr, EP_IntValue) );
      assert( pExpr->u.zToken[0]=='x' || pExpr->u.zToken[0]=='X' );
      assert( pExpr->u.zToken[1]=='\'' );
      z = &pExpr->u.zToken[2];
      n = sqlite3Strlen30(z) - 1;
      assert( z[n]=='\'' );
      zBlob = sqlite3HexToBlob(sqlite3VdbeDb(v), z, n);
      sqlite3VdbeAddOp4(v, OP_Blob, n/2, target, 0, zBlob, P4_DYNAMIC);
      return target;
    }
#endif
    case TK_VARIABLE: {
      assert( !ExprHasProperty(pExpr, EP_IntValue) );
      assert( pExpr->u.zToken!=0 );
      assert( pExpr->u.zToken[0]!=0 );
      sqlite3VdbeAddOp2(v, OP_Variable, pExpr->iColumn, target);
      if( pExpr->u.zToken[1]!=0 ){
        const char *z = sqlite3VListNumToName(pParse->pVList, pExpr-
>iColumn);
        assert( pExpr->u.zToken[0]=='?' || strcmp(pExpr->u.zToken, z)==0
);
        pParse->pVList[0] = 0; /* Indicate VList may no longer be
enlarged */
        sqlite3VdbeAppendP4(v, (char*)z, P4_STATIC);
      }
      return target;
    }
    case TK_REGISTER: {
      return pExpr->iTable;
    }
#ifndef SQLITE_OMIT_CAST
    case TK_CAST: {
      /* Expressions of the form:   CAST(pLeft AS token) */
      inReg = sqlite3ExprCodeTarget(pParse, pExpr->pLeft, target);
```

```
      if( inReg!=target ){
        sqlite3VdbeAddOp2(v, OP_SCopy, inReg, target);
        inReg = target;
      }
      sqlite3VdbeAddOp2(v, OP_Cast, target,
                        sqlite3AffinityType(pExpr->u.zToken, 0));
      return inReg;
    }
#endif /* SQLITE_OMIT_CAST */
    case TK_IS:
    case TK_ISNOT:
      op = (op==TK_IS) ? TK_EQ : TK_NE;
      p5 = SQLITE_NULLEQ;
      /* fall-through */
    case TK_LT:
    case TK_LE:
    case TK_GT:
    case TK_GE:
    case TK_NE:
    case TK_EQ: {
      Expr *pLeft = pExpr->pLeft;
      if( sqlite3ExprIsVector(pLeft) ){
        codeVectorCompare(pParse, pExpr, target, op, p5);
      }else{
        r1 = sqlite3ExprCodeTemp(pParse, pLeft, &regFree1);
        r2 = sqlite3ExprCodeTemp(pParse, pExpr->pRight, &regFree2);
        codeCompare(pParse, pLeft, pExpr->pRight, op,
            r1, r2, inReg, SQLITE_STOREP2 | p5,
            ExprHasProperty(pExpr,EP_Commuted));
        assert(TK_LT==OP_Lt); testcase(op==OP_Lt);
VdbeCoverageIf(v,op==OP_Lt);
        assert(TK_LE==OP_Le); testcase(op==OP_Le);
VdbeCoverageIf(v,op==OP_Le);
        assert(TK_GT==OP_Gt); testcase(op==OP_Gt);
VdbeCoverageIf(v,op==OP_Gt);
        assert(TK_GE==OP_Ge); testcase(op==OP_Ge);
VdbeCoverageIf(v,op==OP_Ge);
        assert(TK_EQ==OP_Eq); testcase(op==OP_Eq);
VdbeCoverageIf(v,op==OP_Eq);
        assert(TK_NE==OP_Ne); testcase(op==OP_Ne);
VdbeCoverageIf(v,op==OP_Ne);
        testcase( regFree1==0 );
        testcase( regFree2==0 );
      }
      break;
    }
    case TK_AND:
    case TK_OR:
    case TK_PLUS:
    case TK_STAR:
    case TK_MINUS:
    case TK_REM:
    case TK_BITAND:
    case TK_BITOR:
```

```
    case TK_SLASH:
    case TK_LSHIFT:
    case TK_RSHIFT:
    case TK_CONCAT: {
      assert( TK_AND==OP_And );            testcase( op==TK_AND );
      assert( TK_OR==OP_Or );              testcase( op==TK_OR );
      assert( TK_PLUS==OP_Add );           testcase( op==TK_PLUS );
      assert( TK_MINUS==OP_Subtract );     testcase( op==TK_MINUS );
      assert( TK_REM==OP_Remainder );      testcase( op==TK_REM );
      assert( TK_BITAND==OP_BitAnd );      testcase( op==TK_BITAND );
      assert( TK_BITOR==OP_BitOr );        testcase( op==TK_BITOR );
      assert( TK_SLASH==OP_Divide );       testcase( op==TK_SLASH );
      assert( TK_LSHIFT==OP_ShiftLeft );   testcase( op==TK_LSHIFT );
      assert( TK_RSHIFT==OP_ShiftRight );  testcase( op==TK_RSHIFT );
      assert( TK_CONCAT==OP_Concat );      testcase( op==TK_CONCAT );
      r1 = sqlite3ExprCodeTemp(pParse, pExpr->pLeft, &regFree1);
      r2 = sqlite3ExprCodeTemp(pParse, pExpr->pRight, &regFree2);
      sqlite3VdbeAddOp3(v, op, r2, r1, target);
      testcase( regFree1==0 );
      testcase( regFree2==0 );
      break;
    }
    case TK_UMINUS: {
      Expr *pLeft = pExpr->pLeft;
      assert( pLeft );
      if( pLeft->op==TK_INTEGER ){
        codeInteger(pParse, pLeft, 1, target);
        return target;
#ifndef SQLITE_OMIT_FLOATING_POINT
      }else if( pLeft->op==TK_FLOAT ){
        assert( !ExprHasProperty(pExpr, EP_IntValue) );
        codeReal(v, pLeft->u.zToken, 1, target);
        return target;
#endif
      }else{
        tempX.op = TK_INTEGER;
        tempX.flags = EP_IntValue|EP_TokenOnly;
        tempX.u.iValue = 0;
        r1 = sqlite3ExprCodeTemp(pParse, &tempX, &regFree1);
        r2 = sqlite3ExprCodeTemp(pParse, pExpr->pLeft, &regFree2);
        sqlite3VdbeAddOp3(v, OP_Subtract, r2, r1, target);
        testcase( regFree2==0 );
      }
      break;
    }
    case TK_BITNOT:
    case TK_NOT: {
      assert( TK_BITNOT==OP_BitNot );   testcase( op==TK_BITNOT );
      assert( TK_NOT==OP_Not );         testcase( op==TK_NOT );
      r1 = sqlite3ExprCodeTemp(pParse, pExpr->pLeft, &regFree1);
      testcase( regFree1==0 );
      sqlite3VdbeAddOp2(v, op, r1, inReg);
      break;
    }
```

```
    case TK_TRUTH: {
      int isTrue;    /* IS TRUE or IS NOT TRUE */
      int bNormal;   /* IS TRUE or IS FALSE */
      r1 = sqlite3ExprCodeTemp(pParse, pExpr->pLeft, &regFree1);
      testcase( regFree1==0 );
      isTrue = sqlite3ExprTruthValue(pExpr->pRight);
      bNormal = pExpr->op2==TK_IS;
      testcase( isTrue && bNormal);
      testcase( !isTrue && bNormal);
      sqlite3VdbeAddOp4Int(v, OP_IsTrue, r1, inReg, !isTrue, isTrue ^
bNormal);
      break;
    }
    case TK_ISNULL:
    case TK_NOTNULL: {
      int addr;
      assert( TK_ISNULL==OP_IsNull );   testcase( op==TK_ISNULL );
      assert( TK_NOTNULL==OP_NotNull ); testcase( op==TK_NOTNULL );
      sqlite3VdbeAddOp2(v, OP_Integer, 1, target);
      r1 = sqlite3ExprCodeTemp(pParse, pExpr->pLeft, &regFree1);
      testcase( regFree1==0 );
      addr = sqlite3VdbeAddOp1(v, op, r1);
      VdbeCoverageIf(v, op==TK_ISNULL);
      VdbeCoverageIf(v, op==TK_NOTNULL);
      sqlite3VdbeAddOp2(v, OP_Integer, 0, target);
      sqlite3VdbeJumpHere(v, addr);
      break;
    }
    case TK_AGG_FUNCTION: {
      AggInfo *pInfo = pExpr->pAggInfo;
      if( pInfo==0 ){
        assert( !ExprHasProperty(pExpr, EP_IntValue) );
        sqlite3ErrorMsg(pParse, "misuse of aggregate: %s()", pExpr-
>u.zToken);
      }else{
        return pInfo->aFunc[pExpr->iAgg].iMem;
      }
      break;
    }
    case TK_FUNCTION: {
      ExprList *pFarg;       /* List of function arguments */
      int nFarg;             /* Number of function arguments */
      FuncDef *pDef;         /* The function definition object */
      const char *zId;       /* The function name */
      u32 constMask = 0;     /* Mask of function arguments that are
constant */
      int i;                 /* Loop counter */
      sqlite3 *db = pParse->db;  /* The database connection */
      u8 enc = ENC(db);      /* The text encoding used by this database
*/
      CollSeq *pColl = 0;    /* A collating sequence */

#ifndef SQLITE_OMIT_WINDOWFUNC
      if( ExprHasProperty(pExpr, EP_WinFunc) ){
```

```
        return pExpr->y.pWin->regResult;
      }
#endif

      if( ConstFactorOk(pParse) && sqlite3ExprIsConstantNotJoin(pExpr) ){
        /* SQL functions can be expensive. So try to move constant
functions
        ** out of the inner loop, even if that means an extra OP_Copy. */
        return sqlite3ExprCodeAtInit(pParse, pExpr, -1);
      }
      assert( !ExprHasProperty(pExpr, EP_xIsSelect) );
      if( ExprHasProperty(pExpr, EP_TokenOnly) ){
        pFarg = 0;
      }else{
        pFarg = pExpr->x.pList;
      }
      nFarg = pFarg ? pFarg->nExpr : 0;
      assert( !ExprHasProperty(pExpr, EP_IntValue) );
      zId = pExpr->u.zToken;
      pDef = sqlite3FindFunction(db, zId, nFarg, enc, 0);
#ifdef SQLITE_ENABLE_UNKNOWN_SQL_FUNCTION
      if( pDef==0 && pParse->explain ){
        pDef = sqlite3FindFunction(db, "unknown", nFarg, enc, 0);
      }
#endif
      if( pDef==0 || pDef->xFinalize!=0 ){
        sqlite3ErrorMsg(pParse, "unknown function: %s()", zId);
        break;
      }

      /* Attempt a direct implementation of the built-in COALESCE() and
      ** IFNULL() functions.  This avoids unnecessary evaluation of
      ** arguments past the first non-NULL argument.
      */
      if( pDef->funcFlags & SQLITE_FUNC_COALESCE ){
        int endCoalesce = sqlite3VdbeMakeLabel(pParse);
        assert( nFarg>=2 );
        sqlite3ExprCode(pParse, pFarg->a[0].pExpr, target);
        for(i=1; i<nFarg; i++){
          sqlite3VdbeAddOp2(v, OP_NotNull, target, endCoalesce);
          VdbeCoverage(v);
          sqlite3ExprCode(pParse, pFarg->a[i].pExpr, target);
        }
        sqlite3VdbeResolveLabel(v, endCoalesce);
        break;
      }

      /* The UNLIKELY() function is a no-op.  The result is the value
      ** of the first argument.
      */
      if( pDef->funcFlags & SQLITE_FUNC_UNLIKELY ){
        assert( nFarg>=1 );
        return sqlite3ExprCodeTarget(pParse, pFarg->a[0].pExpr, target);
      }
```

```
#ifdef SQLITE_DEBUG
      /* The AFFINITY() function evaluates to a string that describes
      ** the type affinity of the argument.  This is used for testing of
      ** the SQLite type logic.
      */
      if( pDef->funcFlags & SQLITE_FUNC_AFFINITY ){
        const char *azAff[] = { "blob", "text", "numeric", "integer",
"real" };
        char aff;
        assert( nFarg==1 );
        aff = sqlite3ExprAffinity(pFarg->a[0].pExpr);
        sqlite3VdbeLoadString(v, target,
                (aff<=SQLITE_AFF_NONE) ? "none" : azAff[aff-
SQLITE_AFF_BLOB]);
        return target;
      }
#endif

      for(i=0; i<nFarg; i++){
        if( i<32 && sqlite3ExprIsConstant(pFarg->a[i].pExpr) ){
          testcase( i==31 );
          constMask |= MASKBIT32(i);
        }
        if( (pDef->funcFlags & SQLITE_FUNC_NEEDCOLL)!=0 && !pColl ){
          pColl = sqlite3ExprCollSeq(pParse, pFarg->a[i].pExpr);
        }
      }
      if( pFarg ){
        if( constMask ){
          r1 = pParse->nMem+1;
          pParse->nMem += nFarg;
        }else{
          r1 = sqlite3GetTempRange(pParse, nFarg);
        }

        /* For length() and typeof() functions with a column argument,
        ** set the P5 parameter to the OP_Column opcode to
OPFLAG_LENGTHARG
        ** or OPFLAG_TYPEOFARG respectively, to avoid unnecessary data
        ** loading.
        */
        if( (pDef->funcFlags &
(SQLITE_FUNC_LENGTH|SQLITE_FUNC_TYPEOF))!=0 ){
          u8 exprOp;
          assert( nFarg==1 );
          assert( pFarg->a[0].pExpr!=0 );
          exprOp = pFarg->a[0].pExpr->op;
          if( exprOp==TK_COLUMN || exprOp==TK_AGG_COLUMN ){
            assert( SQLITE_FUNC_LENGTH==OPFLAG_LENGTHARG );
            assert( SQLITE_FUNC_TYPEOF==OPFLAG_TYPEOFARG );
            testcase( pDef->funcFlags & OPFLAG_LENGTHARG );
            pFarg->a[0].pExpr->op2 =
                  pDef->funcFlags & (OPFLAG_LENGTHARG|OPFLAG_TYPEOFARG);
```

```
          }
        }

        sqlite3ExprCodeExprList(pParse, pFarg, r1, 0,
                                SQLITE_ECEL_DUP|SQLITE_ECEL_FACTOR);
      }else{
        r1 = 0;
      }
#ifndef SQLITE_OMIT_VIRTUALTABLE
      /* Possibly overload the function if the first argument is
      ** a virtual table column.
      **
      ** For infix functions (LIKE, GLOB, REGEXP, and MATCH) use the
      ** second argument, not the first, as the argument to test to
      ** see if it is a column in a virtual table.  This is done because
      ** the left operand of infix functions (the operand we want to
      ** control overloading) ends up as the second argument to the
      ** function.  The expression "A glob B" is equivalent to
      ** "glob(B,A).  We want to use the A in "A glob B" to test
      ** for function overloading.  But we use the B term in "glob(B,A)".
      */
      if( nFarg>=2 && ExprHasProperty(pExpr, EP_InfixFunc) ){
        pDef = sqlite3VtabOverloadFunction(db, pDef, nFarg, pFarg-
>a[1].pExpr);
      }else if( nFarg>0 ){
        pDef = sqlite3VtabOverloadFunction(db, pDef, nFarg, pFarg-
>a[0].pExpr);
      }
#endif
      if( pDef->funcFlags & SQLITE_FUNC_NEEDCOLL ){
        if( !pColl ) pColl = db->pDfltColl;
        sqlite3VdbeAddOp4(v, OP_CollSeq, 0, 0, 0, (char *)pColl,
P4_COLLSEQ);
      }
#ifdef SQLITE_ENABLE_OFFSET_SQL_FUNC
      if( pDef->funcFlags & SQLITE_FUNC_OFFSET ){
        Expr *pArg = pFarg->a[0].pExpr;
        if( pArg->op==TK_COLUMN ){
          sqlite3VdbeAddOp3(v, OP_Offset, pArg->iTable, pArg->iColumn,
target);
        }else{
          sqlite3VdbeAddOp2(v, OP_Null, 0, target);
        }
      }else
#endif
      {
        sqlite3VdbeAddFunctionCall(pParse, constMask, r1, target, nFarg,
                                   pDef, pExpr->op2);
      }
      if( nFarg && constMask==0 ){
        sqlite3ReleaseTempRange(pParse, r1, nFarg);
      }
      return target;
    }
```

```
#ifndef SQLITE_OMIT_SUBQUERY
    case TK_EXISTS:
    case TK_SELECT: {
      int nCol;
      testcase( op==TK_EXISTS );
      testcase( op==TK_SELECT );
      if( op==TK_SELECT && (nCol = pExpr->x.pSelect->pEList->nExpr)!=1 ){
        sqlite3SubselectError(pParse, nCol, 1);
      }else{
        return sqlite3CodeSubselect(pParse, pExpr);
      }
      break;
    }
    case TK_SELECT_COLUMN: {
      int n;
      if( pExpr->pLeft->iTable==0 ){
        pExpr->pLeft->iTable = sqlite3CodeSubselect(pParse, pExpr-
>pLeft);
      }
      assert( pExpr->iTable==0 || pExpr->pLeft->op==TK_SELECT );
      if( pExpr->iTable!=0
       && pExpr->iTable!=(n = sqlite3ExprVectorSize(pExpr->pLeft))
      ){
        sqlite3ErrorMsg(pParse, "%d columns assigned %d values",
                                pExpr->iTable, n);
      }
      return pExpr->pLeft->iTable + pExpr->iColumn;
    }
    case TK_IN: {
      int destIfFalse = sqlite3VdbeMakeLabel(pParse);
      int destIfNull = sqlite3VdbeMakeLabel(pParse);
      sqlite3VdbeAddOp2(v, OP_Null, 0, target);
      sqlite3ExprCodeIN(pParse, pExpr, destIfFalse, destIfNull);
      sqlite3VdbeAddOp2(v, OP_Integer, 1, target);
      sqlite3VdbeResolveLabel(v, destIfFalse);
      sqlite3VdbeAddOp2(v, OP_AddImm, target, 0);
      sqlite3VdbeResolveLabel(v, destIfNull);
      return target;
    }
#endif /* SQLITE_OMIT_SUBQUERY */


    /*
    **    x BETWEEN y AND z
    **
    ** This is equivalent to
    **
    **    x>=y AND x<=z
    **
    ** X is stored in pExpr->pLeft.
    ** Y is stored in pExpr->pList->a[0].pExpr.
    ** Z is stored in pExpr->pList->a[1].pExpr.
    */
    case TK_BETWEEN: {
```

```
      exprCodeBetween(pParse, pExpr, target, 0, 0);
      return target;
    }
    case TK_SPAN:
    case TK_COLLATE:
    case TK_UPLUS: {
      pExpr = pExpr->pLeft;
      goto expr_code_doover; /* 2018-04-28: Prevent deep recursion.
OSSFuzz. */
    }

    case TK_TRIGGER: {
      /* If the opcode is TK_TRIGGER, then the expression is a reference
      ** to a column in the new.* or old.* pseudo-tables available to
      ** trigger programs. In this case Expr.iTable is set to 1 for the
      ** new.* pseudo-table, or 0 for the old.* pseudo-table.
Expr.iColumn
      ** is set to the column of the pseudo-table to read, or to -1 to
      ** read the rowid field.
      **
      ** The expression is implemented using an OP_Param opcode. The p1
      ** parameter is set to 0 for an old.rowid reference, or to (i+1)
      ** to reference another column of the old.* pseudo-table, where
      ** i is the index of the column. For a new.rowid reference, p1 is
      ** set to (n+1), where n is the number of columns in each pseudo-
table.
      ** For a reference to any other column in the new.* pseudo-table,
p1
      ** is set to (n+2+i), where n and i are as defined previously. For
      ** example, if the table on which triggers are being fired is
      ** declared as:
      **
      **   CREATE TABLE t1(a, b);
      **
      ** Then p1 is interpreted as follows:
      **
      **   p1==0   ->    old.rowid     p1==3   ->    new.rowid
      **   p1==1   ->    old.a         p1==4   ->    new.a
      **   p1==2   ->    old.b         p1==5   ->    new.b
      */
      Table *pTab = pExpr->y.pTab;
      int iCol = pExpr->iColumn;
      int p1 = pExpr->iTable * (pTab->nCol+1) + 1
                  + (iCol>=0 ? sqlite3TableColumnToStorage(pTab, iCol)
: -1);

      assert( pExpr->iTable==0 || pExpr->iTable==1 );
      assert( iCol>=-1 && iCol<pTab->nCol );
      assert( pTab->iPKey<0 || iCol!=pTab->iPKey );
      assert( p1>=0 && p1<(pTab->nCol*2+2) );

      sqlite3VdbeAddOp2(v, OP_Param, p1, target);
      VdbeComment((v, "r[%d]=%s.%s", target,
        (pExpr->iTable ? "new" : "old"),
```

```
        (pExpr->iColumn<0 ? "rowid" : pExpr->y.pTab->aCol[iCol].zName)
      ));

#ifndef SQLITE_OMIT_FLOATING_POINT
      /* If the column has REAL affinity, it may currently be stored as
an
      ** integer. Use OP_RealAffinity to make sure it is really real.
      **
      ** EVIDENCE-OF: R-60985-57662 SQLite will convert the value back to
      ** floating point when extracting it from the record.  */
      if( iCol>=0 && pTab->aCol[iCol].affinity==SQLITE_AFF_REAL ){
        sqlite3VdbeAddOp1(v, OP_RealAffinity, target);
      }
#endif
      break;
    }

    case TK_VECTOR: {
      sqlite3ErrorMsg(pParse, "row value misused");
      break;
    }

    /* TK_IF_NULL_ROW Expr nodes are inserted ahead of expressions
    ** that derive from the right-hand table of a LEFT JOIN.  The
    ** Expr.iTable value is the table number for the right-hand table.
    ** The expression is only evaluated if that table is not currently
    ** on a LEFT JOIN NULL row.
    */
    case TK_IF_NULL_ROW: {
      int addrINR;
      u8 okConstFactor = pParse->okConstFactor;
      addrINR = sqlite3VdbeAddOp1(v, OP_IfNullRow, pExpr->iTable);
      /* Temporarily disable factoring of constant expressions, since
      ** even though expressions may appear to be constant, they are not
      ** really constant because they originate from the right-hand side
      ** of a LEFT JOIN. */
      pParse->okConstFactor = 0;
      inReg = sqlite3ExprCodeTarget(pParse, pExpr->pLeft, target);
      pParse->okConstFactor = okConstFactor;
      sqlite3VdbeJumpHere(v, addrINR);
      sqlite3VdbeChangeP3(v, addrINR, inReg);
      break;
    }

    /*
    ** Form A:
    **   CASE x WHEN e1 THEN r1 WHEN e2 THEN r2 ... WHEN eN THEN rN ELSE
y END
    **
    ** Form B:
    **   CASE WHEN e1 THEN r1 WHEN e2 THEN r2 ... WHEN eN THEN rN ELSE y
END
    **
```

```
    ** Form A is can be transformed into the equivalent form B as
follows:
    **   CASE WHEN x=e1 THEN r1 WHEN x=e2 THEN r2 ...
    **        WHEN x=eN THEN rN ELSE y END
    **
    ** X (if it exists) is in pExpr->pLeft.
    ** Y is in the last element of pExpr->x.pList if pExpr->x.pList-
>nExpr is
    ** odd.  The Y is also optional.  If the number of elements in
x.pList
    ** is even, then Y is omitted and the "otherwise" result is NULL.
    ** Ei is in pExpr->pList->a[i*2] and Ri is pExpr->pList->a[i*2+1].
    **
    ** The result of the expression is the Ri for the first matching Ei,
    ** or if there is no matching Ei, the ELSE term Y, or if there is
    ** no ELSE term, NULL.
    */
    default: assert( op==TK_CASE ); {
      int endLabel;                     /* GOTO label for end of CASE
stmt */
      int nextCase;                     /* GOTO label for next WHEN
clause */
      int nExpr;                        /* 2x number of WHEN terms */
      int i;                            /* Loop counter */
      ExprList *pEList;                 /* List of WHEN terms */
      struct ExprList_item *aListelem;  /* Array of WHEN terms */
      Expr opCompare;                   /* The X==Ei expression */
      Expr *pX;                         /* The X expression */
      Expr *pTest = 0;                  /* X==Ei (form A) or just Ei
(form B) */
      Expr *pDel = 0;
      sqlite3 *db = pParse->db;

      assert( !ExprHasProperty(pExpr, EP_xIsSelect) && pExpr->x.pList );
      assert(pExpr->x.pList->nExpr > 0);
      pEList = pExpr->x.pList;
      aListelem = pEList->a;
      nExpr = pEList->nExpr;
      endLabel = sqlite3VdbeMakeLabel(pParse);
      if( (pX = pExpr->pLeft)!=0 ){
        pDel = sqlite3ExprDup(db, pX, 0);
        if( db->mallocFailed ){
          sqlite3ExprDelete(db, pDel);
          break;
        }
        testcase( pX->op==TK_COLUMN );
        exprToRegister(pDel, exprCodeVector(pParse, pDel, &regFree1));
        testcase( regFree1==0 );
        memset(&opCompare, 0, sizeof(opCompare));
        opCompare.op = TK_EQ;
        opCompare.pLeft = pDel;
        pTest = &opCompare;
        /* Ticket b351d95f9cd5ef17e9d9dbae18f5ca8611190001:
        ** The value in regFree1 might get SCopy-ed into the file result.
```

```
        ** So make sure that the regFree1 register is not reused for
other
        ** purposes and possibly overwritten.  */
        regFree1 = 0;
      }
      for(i=0; i<nExpr-1; i=i+2){
        if( pX ){
          assert( pTest!=0 );
          opCompare.pRight = aListelem[i].pExpr;
        }else{
          pTest = aListelem[i].pExpr;
        }
        nextCase = sqlite3VdbeMakeLabel(pParse);
        testcase( pTest->op==TK_COLUMN );
        sqlite3ExprIfFalse(pParse, pTest, nextCase, SQLITE_JUMPIFNULL);
        testcase( aListelem[i+1].pExpr->op==TK_COLUMN );
        sqlite3ExprCode(pParse, aListelem[i+1].pExpr, target);
        sqlite3VdbeGoto(v, endLabel);
        sqlite3VdbeResolveLabel(v, nextCase);
      }
      if( (nExpr&1)!=0 ){
        sqlite3ExprCode(pParse, pEList->a[nExpr-1].pExpr, target);
      }else{
        sqlite3VdbeAddOp2(v, OP_Null, 0, target);
      }
      sqlite3ExprDelete(db, pDel);
      sqlite3VdbeResolveLabel(v, endLabel);
      break;
    }
#ifndef SQLITE_OMIT_TRIGGER
    case TK_RAISE: {
      assert( pExpr->affExpr==OE_Rollback
           || pExpr->affExpr==OE_Abort
           || pExpr->affExpr==OE_Fail
           || pExpr->affExpr==OE_Ignore
      );
      if( !pParse->pTriggerTab ){
        sqlite3ErrorMsg(pParse,
                        "RAISE() may only be used within a trigger-
program");
        return 0;
      }
      if( pExpr->affExpr==OE_Abort ){
        sqlite3MayAbort(pParse);
      }
      assert( !ExprHasProperty(pExpr, EP_IntValue) );
      if( pExpr->affExpr==OE_Ignore ){
        sqlite3VdbeAddOp4(
            v, OP_Halt, SQLITE_OK, OE_Ignore, 0, pExpr->u.zToken,0);
        VdbeCoverage(v);
      }else{
        sqlite3HaltConstraint(pParse, SQLITE_CONSTRAINT_TRIGGER,
                              pExpr->affExpr, pExpr->u.zToken, 0, 0);
      }
```

```
      break;
    }
#endif
  }
  sqlite3ReleaseTempReg(pParse, regFree1);
  sqlite3ReleaseTempReg(pParse, regFree2);
  return inReg;
}
<sep>
static struct scatterlist *alloc_sgtable(int size)
{
      int alloc_size, nents, i;
      struct page *new_page;
      struct scatterlist *iter;
      struct scatterlist *table;

      nents = DIV_ROUND_UP(size, PAGE_SIZE);
      table = kcalloc(nents, sizeof(*table), GFP_KERNEL);
      if (!table)
            return NULL;
      sg_init_table(table, nents);
      iter = table;
      for_each_sg(table, iter, sg_nents(table), i) {
            new_page = alloc_page(GFP_KERNEL);
            if (!new_page) {
                  /* release all previous allocated pages in the table */
                  iter = table;
                  for_each_sg(table, iter, sg_nents(table), i) {
                        new_page = sg_page(iter);
                        if (new_page)
                              __free_page(new_page);
                  }
                  return NULL;
            }
            alloc_size = min_t(int, size, PAGE_SIZE);
            size -= PAGE_SIZE;
            sg_set_page(iter, new_page, alloc_size, 0);
      }
      return table;
}
<sep>
static ssize_t print_cpu_modalias(struct device *dev,
                          struct device_attribute *attr,
                          char *buf)
{
      ssize_t n;
      u32 i;

      n = sprintf(buf, "cpu:type:" CPU_FEATURE_TYPEFMT ":feature:",
                CPU_FEATURE_TYPEVAL);

      for (i = 0; i < MAX_CPU_FEATURES; i++)
            if (cpu_have_feature(i)) {
```

```
                if (PAGE_SIZE < n + sizeof(",XXXX\n")) {
                      WARN(1, "CPU features overflow page\n");
                      break;
                }
                n += sprintf(&buf[n], ",%04X", i);
          }
      buf[n++] = '\n';
      return n;
}
<sep>
  bool ms_verify_authorizer(Connection *con, int peer_type, int protocol,
                            bufferlist& authorizer, bufferlist&
authorizer_reply,
                            bool& isvalid, CryptoKey& session_key)
override {
    isvalid = true;
    return true;
  }
<sep>
int nft_parse_register_store(const struct nft_ctx *ctx,
                      const struct nlattr *attr, u8 *dreg,
                      const struct nft_data *data,
                      enum nft_data_types type, unsigned int len)
{
      int err;
      u32 reg;

      reg = nft_parse_register(attr);
      err = nft_validate_register_store(ctx, reg, data, type, len);
      if (err < 0)
            return err;

      *dreg = reg;
      return 0;
}
<sep>
static int idprime_get_token_name(sc_card_t* card, char** tname)
{
      idprime_private_data_t * priv = card->drv_data;
      sc_path_t tinfo_path = {"\x00\x00", 2, 0, 0, SC_PATH_TYPE_PATH,
{"", 0}};
      sc_file_t *file = NULL;
      u8 buf[2];
      int r;

      LOG_FUNC_CALLED(card->ctx);

      if (tname == NULL) {
            LOG_FUNC_RETURN(card->ctx, SC_ERROR_INVALID_ARGUMENTS);
      }

      if (!priv->tinfo_present) {
            LOG_FUNC_RETURN(card->ctx, SC_ERROR_NOT_SUPPORTED);
      }
```

```
      memcpy(tinfo_path.value, priv->tinfo_df, 2);
      r = iso_ops->select_file(card, &tinfo_path, &file);
      if (r != SC_SUCCESS || file->size == 0) {
            sc_file_free(file);
            LOG_FUNC_RETURN(card->ctx, SC_ERROR_NOT_SUPPORTED);
      }

      /* First two bytes lists 0x01, the second indicates length */
      r = iso_ops->read_binary(card, 0, buf, 2, 0);
      if (r < 2 || buf[1] > file->size) { /* make sure we do not overrun
*/
            sc_file_free(file);
            LOG_FUNC_RETURN(card->ctx, r);
      }
      sc_file_free(file);

      *tname = malloc(buf[1]);
      if (*tname == NULL) {
            LOG_FUNC_RETURN(card->ctx, SC_ERROR_OUT_OF_MEMORY);
      }

      r = iso_ops->read_binary(card, 2, (unsigned char *)*tname, buf[1],
0);
      if (r < 1) {
            free(*tname);
            LOG_FUNC_RETURN(card->ctx, r);
      }

      if ((*tname)[r-1] != '\0') {
            (*tname)[r-1] = '\0';
      }
      LOG_FUNC_RETURN(card->ctx, SC_SUCCESS);
}
<sep>
Status RoleGraph::getBSONForRole(RoleGraph* graph,
                                 const RoleName& roleName,
                                 mutablebson::Element result) try {
    if (!graph->roleExists(roleName)) {
        return Status(ErrorCodes::RoleNotFound,
                      str::stream() << roleName.getFullName() << "does
not name an existing role");
    }
    std::string id = str::stream() << roleName.getDB() << "." <<
roleName.getRole();
    uassertStatusOK(result.appendString("_id", id));
    uassertStatusOK(
        result.appendString(AuthorizationManager::ROLE_NAME_FIELD_NAME,
roleName.getRole()));
    uassertStatusOK(
        result.appendString(AuthorizationManager::ROLE_DB_FIELD_NAME,
roleName.getDB()));

    // Build privileges array
```

```
    mutablebson::Element privilegesArrayElement =
        result.getDocument().makeElementArray("privileges");
    uassertStatusOK(result.pushBack(privilegesArrayElement));
    const PrivilegeVector& privileges = graph-
>getDirectPrivileges(roleName);
    uassertStatusOK(Privilege::getBSONForPrivileges(privileges,
privilegesArrayElement));

    // Build roles array
    mutablebson::Element rolesArrayElement =
result.getDocument().makeElementArray("roles");
    uassertStatusOK(result.pushBack(rolesArrayElement));
    for (RoleNameIterator roles = graph->getDirectSubordinates(roleName);
roles.more();
         roles.next()) {
        const RoleName& subRole = roles.get();
        mutablebson::Element roleObj =
result.getDocument().makeElementObject("");
        uassertStatusOK(

roleObj.appendString(AuthorizationManager::ROLE_NAME_FIELD_NAME,
subRole.getRole()));
        uassertStatusOK(

roleObj.appendString(AuthorizationManager::ROLE_DB_FIELD_NAME,
subRole.getDB()));
        uassertStatusOK(rolesArrayElement.pushBack(roleObj));
    }

    return Status::OK();
} catch (...) {
<sep>
  void ComputeAsync(OpKernelContext* c, DoneCallback done) override {
    auto col_params = new CollectiveParams();
    auto done_with_cleanup = [col_params, done = std::move(done)]() {
      done();
      col_params->Unref();
    };
    core::RefCountPtr<CollectiveGroupResource> resource;
    OP_REQUIRES_OK_ASYNC(c, LookupResource(c, HandleFromInput(c, 1),
&resource),
                         done);

    Tensor group_assignment = c->input(2);

    OP_REQUIRES_OK_ASYNC(
        c,
        FillCollectiveParams(col_params, group_assignment,
                             ALL_TO_ALL_COLLECTIVE, resource.get()),
        done);
    col_params->instance.shape = c->input(0).shape();
    VLOG(1) << "CollectiveAllToAll group_size " << col_params-
>group.group_size
```

```
            << " group_key " << col_params->group.group_key << "
instance_key "
            << col_params->instance.instance_key;
    // Allocate the output tensor, trying to reuse the input.
    Tensor* output = nullptr;
    OP_REQUIRES_OK_ASYNC(c,
                         c->forward_input_or_allocate_output(
                             {0}, 0, col_params->instance.shape,
&output),
                         done_with_cleanup);
    Run(c, col_params, std::move(done_with_cleanup));
  }
<sep>
LIBOPENMPT_MODPLUG_API unsigned int ModPlug_SampleName(ModPlugFile* file,
unsigned int qual, char* buff)
{
      const char* str;
      unsigned int retval;
      size_t tmpretval;
      if(!file) return 0;
      str = openmpt_module_get_sample_name(file->mod,qual-1);
      if(!str){
            if(buff){
                  *buff = '\0';
            }
            return 0;
      }
      tmpretval = strlen(str);
      if(tmpretval>=INT_MAX){
            tmpretval = INT_MAX-1;
      }
      retval = (int)tmpretval;
      if(buff){
            memcpy(buff,str,retval+1);
            buff[retval] = '\0';
      }
      openmpt_free_string(str);
      return retval;
}
<sep>
read_yin_rpc_action(struct lys_module *module, struct lys_node *parent,
struct lyxml_elem *yin,
                    int options, struct unres_schema *unres)
{
    struct ly_ctx *ctx = module->ctx;
    struct lyxml_elem *sub, *next, root;
    struct lys_node *node = NULL;
    struct lys_node *retval;
    struct lys_node_rpc_action *rpc;
    int r;
    int c_tpdf = 0, c_ftrs = 0, c_input = 0, c_output = 0, c_ext = 0;
    void *reallocated;

    if (!strcmp(yin->name, "action") && (module->version < 2)) {
```

```
        LOGVAL(ctx, LYE_INSTMT, LY_VLOG_LYS, parent, "action");
        return NULL;
    }

    /* init */
    memset(&root, 0, sizeof root);

    rpc = calloc(1, sizeof *rpc);
    LY_CHECK_ERR_RETURN(!rpc, LOGMEM(ctx), NULL);

    rpc->nodetype = (!strcmp(yin->name, "rpc") ? LYS_RPC : LYS_ACTION);
    rpc->prev = (struct lys_node *)rpc;
    retval = (struct lys_node *)rpc;

    if (read_yin_common(module, parent, retval, LYEXT_PAR_NODE, yin,
OPT_IDENT | OPT_MODULE, unres)) {
        goto error;
    }

    LOGDBG(LY_LDGYIN, "parsing %s statement \"%s\"", yin->name, retval-
>name);

    /* insert the node into the schema tree */
    if (lys_node_addchild(parent, lys_main_module(module), retval,
options)) {
        goto error;
    }

    /* process rpc's specific children */
    LY_TREE_FOR_SAFE(yin->child, next, sub) {
        if (strcmp(sub->ns->value, LY_NSYIN)) {
            /* extension */
            YIN_CHECK_ARRAY_OVERFLOW_GOTO(ctx, c_ext, retval->ext_size,
"extensions",
                                          rpc->nodetype == LYS_RPC ?
"rpc" : "action", error);
            c_ext++;
            continue;
        } else if (!strcmp(sub->name, "input")) {
            if (c_input) {
                LOGVAL(ctx, LYE_TOOMANY, LY_VLOG_LYS, retval, sub->name,
yin->name);
                goto error;
            }
            c_input++;
            lyxml_unlink_elem(ctx, sub, 2);
            lyxml_add_child(ctx, &root, sub);
        } else if (!strcmp(sub->name, "output")) {
            if (c_output) {
                LOGVAL(ctx, LYE_TOOMANY, LY_VLOG_LYS, retval, sub->name,
yin->name);
                goto error;
            }
            c_output++;
```

```
            lyxml_unlink_elem(ctx, sub, 2);
            lyxml_add_child(ctx, &root, sub);

            /* data statements */
        } else if (!strcmp(sub->name, "grouping")) {
            lyxml_unlink_elem(ctx, sub, 2);
            lyxml_add_child(ctx, &root, sub);

            /* array counters */
        } else if (!strcmp(sub->name, "typedef")) {
            YIN_CHECK_ARRAY_OVERFLOW_GOTO(ctx, c_tpdf, rpc->tpdf_size,
"typedefs",
                                          rpc->nodetype == LYS_RPC ?
"rpc" : "action", error);
            c_tpdf++;
        } else if (!strcmp(sub->name, "if-feature")) {
            YIN_CHECK_ARRAY_OVERFLOW_GOTO(ctx, c_ftrs, retval-
>iffeature_size, "if-features",
                                          rpc->nodetype == LYS_RPC ?
"rpc" : "action", error);
            c_ftrs++;
        } else {
            LOGVAL(ctx, LYE_INSTMT, LY_VLOG_LYS, retval, sub->name);
            goto error;
        }
    }

    /* middle part - process nodes with cardinality of 0..n except the
data nodes */
    if (c_tpdf) {
        rpc->tpdf = calloc(c_tpdf, sizeof *rpc->tpdf);
        LY_CHECK_ERR_GOTO(!rpc->tpdf, LOGMEM(ctx), error);
    }
    if (c_ftrs) {
        rpc->iffeature = calloc(c_ftrs, sizeof *rpc->iffeature);
        LY_CHECK_ERR_GOTO(!rpc->iffeature, LOGMEM(ctx), error);
    }
    if (c_ext) {
        /* some extensions may be already present from the substatements
*/
        reallocated = realloc(retval->ext, (c_ext + retval->ext_size) *
sizeof *retval->ext);
        LY_CHECK_ERR_GOTO(!reallocated, LOGMEM(ctx), error);
        retval->ext = reallocated;

        /* init memory */
        memset(&retval->ext[retval->ext_size], 0, c_ext * sizeof *retval-
>ext);
    }

    LY_TREE_FOR_SAFE(yin->child, next, sub) {
        if (strcmp(sub->ns->value, LY_NSYIN)) {
            /* extension */
```

```
            r = lyp_yin_fill_ext(retval, LYEXT_PAR_NODE, 0, 0, module,
sub, &retval->ext, &retval->ext_size, unres);
            if (r) {
                goto error;
            }
        } else if (!strcmp(sub->name, "typedef")) {
            r = fill_yin_typedef(module, retval, sub, &rpc->tpdf[rpc-
>tpdf_size], unres);
            rpc->tpdf_size++;
            if (r) {
                goto error;
            }
        } else if (!strcmp(sub->name, "if-feature")) {
            r = fill_yin_iffeature(retval, 0, sub, &rpc->iffeature[rpc-
>iffeature_size], unres);
            rpc->iffeature_size++;
            if (r) {
                goto error;
            }
        }
    }

    lyp_reduce_ext_list(&retval->ext, retval->ext_size, c_ext + retval-
>ext_size);

    /* last part - process data nodes */
    LY_TREE_FOR_SAFE(root.child, next, sub) {
        if (!strcmp(sub->name, "grouping")) {
            node = read_yin_grouping(module, retval, sub, options,
unres);
        } else if (!strcmp(sub->name, "input") || !strcmp(sub->name,
"output")) {
            node = read_yin_input_output(module, retval, sub, options,
unres);
        }
        if (!node) {
            goto error;
        }

        lyxml_free(ctx, sub);
    }

    return retval;

error:
    lys_node_free(retval, NULL, 0);
    while (root.child) {
        lyxml_free(ctx, root.child);
    }
    return NULL;
}
<sep>
getsize_gnutar(
    dle_t      *dle,
```

```
    int          level,
    time_t  dumpsince,
    char      **errmsg)
{
    int pipefd = -1, nullfd = -1;
    pid_t dumppid;
    off_t size = (off_t)-1;
    FILE *dumpout = NULL;
    char *incrname = NULL;
    char *basename = NULL;
    char *dirname = NULL;
    char *inputname = NULL;
    FILE *in = NULL;
    FILE *out = NULL;
    char *line = NULL;
    char *cmd = NULL;
    char *command = NULL;
    char dumptimestr[80];
    struct tm *gmtm;
    int nb_exclude = 0;
    int nb_include = 0;
    GPtrArray *argv_ptr = g_ptr_array_new();
    char *file_exclude = NULL;
    char *file_include = NULL;
    times_t start_time;
    int infd, outfd;
    ssize_t nb;
    char buf[32768];
    char *qdisk = quote_string(dle->disk);
    char *gnutar_list_dir;
    amwait_t wait_status;
    char tmppath[PATH_MAX];

    if (level > 9)
      return -2; /* planner will not even consider this level */

    if(dle->exclude_file) nb_exclude += dle->exclude_file->nb_element;
    if(dle->exclude_list) nb_exclude += dle->exclude_list->nb_element;
    if(dle->include_file) nb_include += dle->include_file->nb_element;
    if(dle->include_list) nb_include += dle->include_list->nb_element;

    if(nb_exclude > 0) file_exclude = build_exclude(dle, 0);
    if(nb_include > 0) file_include = build_include(dle, 0);

    gnutar_list_dir = getconf_str(CNF_GNUTAR_LIST_DIR);
    if (strlen(gnutar_list_dir) == 0)
      gnutar_list_dir = NULL;
    if (gnutar_list_dir) {
      char number[NUM_STR_SIZE];
      int baselevel;
      char *sdisk = sanitise_filename(dle->disk);

      basename = vstralloc(gnutar_list_dir,
                           "/",
```

```
                    g_options->hostname,
                    sdisk,
                    NULL);
        amfree(sdisk);

        g_snprintf(number, SIZEOF(number), "%d", level);
        incrname = vstralloc(basename, "_", number, ".new", NULL);
        unlink(incrname);

        /*
         * Open the listed incremental file from the previous level.
Search
         * backward until one is found.  If none are found (which will also
         * be true for a level 0), arrange to read from /dev/null.
         */
        baselevel = level;
        infd = -1;
        while (infd == -1) {
            if (--baselevel >= 0) {
              g_snprintf(number, SIZEOF(number), "%d", baselevel);
              inputname = newvstralloc(inputname,
                               basename, "_", number, NULL);
            } else {
              inputname = newstralloc(inputname, "/dev/null");
            }
            if ((infd = open(inputname, O_RDONLY)) == -1) {

              *errmsg = vstrallocf(_("gnutar: error opening %s: %s"),
                             inputname, strerror(errno));
              dbprintf("%s\n", *errmsg);
              if (baselevel < 0) {
                  goto common_exit;
              }
              amfree(*errmsg);
            }
        }

        /*
         * Copy the previous listed incremental file to the new one.
         */
        if ((outfd = open(incrname, O_WRONLY|O_CREAT, 0600)) == -1) {
            *errmsg = vstrallocf(_("opening %s: %s"),
                            incrname, strerror(errno));
            dbprintf("%s\n", *errmsg);
            goto common_exit;
        }

        while ((nb = read(infd, &buf, SIZEOF(buf))) > 0) {
            if (full_write(outfd, &buf, (size_t)nb) < (size_t)nb) {
              *errmsg = vstrallocf(_("writing to %s: %s"),
                             incrname, strerror(errno));
              dbprintf("%s\n", *errmsg);
              goto common_exit;
            }
```

```
      }

      if (nb < 0) {
          *errmsg = vstrallocf(_("reading from %s: %s"),
                               inputname, strerror(errno));
          dbprintf("%s\n", *errmsg);
          goto common_exit;
      }

      if (close(infd) != 0) {
          *errmsg = vstrallocf(_("closing %s: %s"),
                               inputname, strerror(errno));
          dbprintf("%s\n", *errmsg);
          goto common_exit;
      }
      if (close(outfd) != 0) {
          *errmsg = vstrallocf(_("closing %s: %s"),
                               incrname, strerror(errno));
          dbprintf("%s\n", *errmsg);
          goto common_exit;
      }

      amfree(inputname);
      amfree(basename);
    }

    gmtm = gmtime(&dumpsince);
    g_snprintf(dumptimestr, SIZEOF(dumptimestr),
           "%04d-%02d-%02d %2d:%02d:%02d GMT",
           gmtm->tm_year + 1900, gmtm->tm_mon+1, gmtm->tm_mday,
           gmtm->tm_hour, gmtm->tm_min, gmtm->tm_sec);

    dirname = amname_to_dirname(dle->device);

    cmd = vstralloc(amlibexecdir, "/", "runtar", NULL);
    g_ptr_array_add(argv_ptr, stralloc("runtar"));
    if (g_options->config)
      g_ptr_array_add(argv_ptr, stralloc(g_options->config));
    else
      g_ptr_array_add(argv_ptr, stralloc("NOCONFIG"));

#ifdef GNUTAR
    g_ptr_array_add(argv_ptr, stralloc(GNUTAR));
#else
    g_ptr_array_add(argv_ptr, stralloc("tar"));
#endif
    g_ptr_array_add(argv_ptr, stralloc("--create"));
    g_ptr_array_add(argv_ptr, stralloc("--file"));
    g_ptr_array_add(argv_ptr, stralloc("/dev/null"));
    /* use --numeric-owner for estimates, to reduce the number of
user/group
     * lookups required */
    g_ptr_array_add(argv_ptr, stralloc("--numeric-owner"));
    g_ptr_array_add(argv_ptr, stralloc("--directory"));
```

```
    canonicalize_pathname(dirname, tmppath);
    g_ptr_array_add(argv_ptr, stralloc(tmppath));
    g_ptr_array_add(argv_ptr, stralloc("--one-file-system"));
    if (gnutar_list_dir) {
      g_ptr_array_add(argv_ptr, stralloc("--listed-incremental"));
      g_ptr_array_add(argv_ptr, stralloc(incrname));
    } else {
      g_ptr_array_add(argv_ptr, stralloc("--incremental"));
      g_ptr_array_add(argv_ptr, stralloc("--newer"));
      g_ptr_array_add(argv_ptr, stralloc(dumptimestr));
    }
#ifdef ENABLE_GNUTAR_ATIME_PRESERVE
    /* --atime-preserve causes gnutar to call
     * utime() after reading files in order to
     * adjust their atime.  However, utime()
     * updates the file's ctime, so incremental
     * dumps will think the file has changed. */
    g_ptr_array_add(argv_ptr, stralloc("--atime-preserve"));
#endif
    g_ptr_array_add(argv_ptr, stralloc("--sparse"));
    g_ptr_array_add(argv_ptr, stralloc("--ignore-failed-read"));
    g_ptr_array_add(argv_ptr, stralloc("--totals"));

    if(file_exclude) {
      g_ptr_array_add(argv_ptr, stralloc("--exclude-from"));
      g_ptr_array_add(argv_ptr, stralloc(file_exclude));
    }

    if(file_include) {
      g_ptr_array_add(argv_ptr, stralloc("--files-from"));
      g_ptr_array_add(argv_ptr, stralloc(file_include));
    }
    else {
      g_ptr_array_add(argv_ptr, stralloc("."));
    }
    g_ptr_array_add(argv_ptr, NULL);

    start_time = curclock();

    if ((nullfd = open("/dev/null", O_RDWR)) == -1) {
      *errmsg = vstrallocf(_("Cannot access /dev/null : %s"),
                     strerror(errno));
      dbprintf("%s\n", *errmsg);
      goto common_exit;
    }

    command = (char *)g_ptr_array_index(argv_ptr, 0);
    dumppid = pipespawnv(cmd, STDERR_PIPE, 0,
                  &nullfd, &nullfd, &pipefd, (char **)argv_ptr->pdata);

    dumpout = fdopen(pipefd,"r");
    if (!dumpout) {
      error(_("Can't fdopen: %s"), strerror(errno));
      /*NOTREACHED*/
```

```
}

for(size = (off_t)-1; (line = agets(dumpout)) != NULL; free(line)) {
  if (line[0] == '\0')
      continue;
  dbprintf("%s\n", line);
  size = handle_dumpline(line);
  if(size > (off_t)-1) {
      amfree(line);
      while ((line = agets(dumpout)) != NULL) {
        if (line[0] != '\0') {
            break;
        }
        amfree(line);
      }
      if (line != NULL) {
        dbprintf("%s\n", line);
        break;
      }
      break;
  }
}
amfree(line);

dbprintf(".....\n");
dbprintf(_("estimate time for %s level %d: %s\n"),
        qdisk,
        level,
        walltime_str(timessub(curclock(), start_time)));
if(size == (off_t)-1) {
  *errmsg = vstrallocf(_("no size line match in %s output"),
                       command);
  dbprintf(_("%s for %s\n"), *errmsg, qdisk);
  dbprintf(".....\n");
} else if(size == (off_t)0 && level == 0) {
  dbprintf(_("possible %s problem -- is \"%s\" really empty?\n"),
          command, dle->disk);
  dbprintf(".....\n");
}
dbprintf(_("estimate size for %s level %d: %lld KB\n"),
        qdisk,
        level,
        (long long)size);

kill(-dumppid, SIGTERM);

dbprintf(_("waiting for %s \"%s\" child\n"),
       command, qdisk);
waitpid(dumppid, &wait_status, 0);
if (WIFSIGNALED(wait_status)) {
  *errmsg = vstrallocf(_("%s terminated with signal %d: see %s"),
                       cmd, WTERMSIG(wait_status), dbfn());
} else if (WIFEXITED(wait_status)) {
  if (WEXITSTATUS(wait_status) != 0) {
```

```
            *errmsg = vstrallocf(_("%s exited with status %d: see %s"),
                                 cmd, WEXITSTATUS(wait_status), dbfn());
          } else {
            /* Normal exit */
          }
        } else {
          *errmsg = vstrallocf(_("%s got bad exit: see %s"),
                               cmd, dbfn());
        }
        dbprintf(_("after %s %s wait\n"), command, qdisk);

    common_exit:

        if (incrname) {
          unlink(incrname);
        }
        amfree(incrname);
        amfree(basename);
        amfree(dirname);
        amfree(inputname);
        g_ptr_array_free_full(argv_ptr);
        amfree(qdisk);
        amfree(cmd);
        amfree(file_exclude);
        amfree(file_include);

        aclose(nullfd);
        afclose(dumpout);
        afclose(in);
        afclose(out);

        return size;
    }
    <sep>
    theme_adium_append_message (EmpathyChatView *view,
                           EmpathyMessage  *msg)
    {
          EmpathyThemeAdium     *theme = EMPATHY_THEME_ADIUM (view);
          EmpathyThemeAdiumPriv *priv = GET_PRIV (theme);
          EmpathyContact        *sender;
          TpMessage             *tp_msg;
          TpAccount             *account;
          gchar                 *body_escaped;
          const gchar           *name;
          const gchar           *contact_id;
          EmpathyAvatar         *avatar;
          const gchar           *avatar_filename = NULL;
          gint64                 timestamp;
          const gchar           *html = NULL;
          const gchar           *func;
          const gchar           *service_name;
          GString               *message_classes = NULL;
          gboolean               is_backlog;
          gboolean               consecutive;
```

```
      gboolean               action;

      if (priv->pages_loading != 0) {
            queue_item (&priv->message_queue, QUEUED_MESSAGE, msg, NULL);
            return;
      }

      /* Get information */
      sender = empathy_message_get_sender (msg);
      account = empathy_contact_get_account (sender);
      service_name = empathy_protocol_name_to_display_name
            (tp_account_get_protocol (account));
      if (service_name == NULL)
            service_name = tp_account_get_protocol (account);
      timestamp = empathy_message_get_timestamp (msg);
      body_escaped = theme_adium_parse_body (theme,
            empathy_message_get_body (msg),
            empathy_message_get_token (msg));
      name = empathy_contact_get_logged_alias (sender);
      contact_id = empathy_contact_get_id (sender);
      action = (empathy_message_get_tptype (msg) ==
TP_CHANNEL_TEXT_MESSAGE_TYPE_ACTION);

      /* If this is a /me probably */
      if (action) {
            gchar *str;

            if (priv->data->version >= 4 || !priv->data->custom_template)
{
                  str = g_strdup_printf ("<span
class='actionMessageUserName'>%s</span>"
                                         "<span
class='actionMessageBody'>%s</span>",
                                         name, body_escaped);
            } else {
                  str = g_strdup_printf ("*%s*", body_escaped);
            }
            g_free (body_escaped);
            body_escaped = str;
      }

      /* Get the avatar filename, or a fallback */
      avatar = empathy_contact_get_avatar (sender);
      if (avatar) {
            avatar_filename = avatar->filename;
      }
      if (!avatar_filename) {
            if (empathy_contact_is_user (sender)) {
                  avatar_filename = priv->data-
>default_outgoing_avatar_filename;
            } else {
                  avatar_filename = priv->data-
>default_incoming_avatar_filename;
            }
```

```
            if (!avatar_filename) {
                  if (!priv->data->default_avatar_filename) {
                        priv->data->default_avatar_filename =
                              empathy_filename_from_icon_name
(EMPATHY_IMAGE_AVATAR_DEFAULT,

GTK_ICON_SIZE_DIALOG);
                  }
                  avatar_filename = priv->data->default_avatar_filename;
            }
      }

      /* We want to join this message with the last one if
       * - senders are the same contact,
       * - last message was recieved recently,
       * - last message and this message both are/aren't backlog, and
       * - DisableCombineConsecutive is not set in theme's settings */
      is_backlog = empathy_message_is_backlog (msg);
      consecutive = empathy_contact_equal (priv->last_contact, sender) &&
            (timestamp - priv->last_timestamp < MESSAGE_JOIN_PERIOD) &&
            (is_backlog == priv->last_is_backlog) &&
            !tp_asv_get_boolean (priv->data->info,
                           "DisableCombineConsecutive", NULL);

      /* Define message classes */
      message_classes = g_string_new ("message");
      if (!priv->has_focus && !is_backlog) {
            if (!priv->has_unread_message) {
                  g_string_append (message_classes, " firstFocus");
                  priv->has_unread_message = TRUE;
            }
            g_string_append (message_classes, " focus");
      }
      if (is_backlog) {
            g_string_append (message_classes, " history");
      }
      if (consecutive) {
            g_string_append (message_classes, " consecutive");
      }
      if (empathy_contact_is_user (sender)) {
            g_string_append (message_classes, " outgoing");
      } else {
            g_string_append (message_classes, " incoming");
      }
      if (empathy_message_should_highlight (msg)) {
            g_string_append (message_classes, " mention");
      }
      if (empathy_message_get_tptype (msg) ==
TP_CHANNEL_TEXT_MESSAGE_TYPE_AUTO_REPLY) {
            g_string_append (message_classes, " autoreply");
      }
      if (action) {
            g_string_append (message_classes, " action");
      }
```

```
	/* FIXME: other classes:
	 * status - the message is a status change
	 * event - the message is a notification of something happening
	 *         (for example, encryption being turned on)
	 * %status% - See %status% in theme_adium_append_html ()
	 */

	/* This is slightly a hack, but it's the only way to add
	 * arbitrary data to messages in the HTML. We add another
	 * class called "x-empathy-message-id-*" to the message. This
	 * way, we can remove the unread marker for this specific
	 * message later. */
	tp_msg = empathy_message_get_tp_message (msg);
	if (tp_msg != NULL) {
		guint32 id;
		gboolean valid;

		id = tp_message_get_pending_message_id (tp_msg, &valid);
		if (valid) {
			g_string_append_printf (message_classes,
			    " x-empathy-message-id-%u", id);
		}
	}

	/* Define javascript function to use */
	if (consecutive) {
		func = priv->allow_scrolling ? "appendNextMessage" :
"appendNextMessageNoScroll";
	} else {
		func = priv->allow_scrolling ? "appendMessage" :
"appendMessageNoScroll";
	}

	if (empathy_contact_is_user (sender)) {
		/* out */
		if (is_backlog) {
			/* context */
			html = consecutive ? priv->data->out_nextcontext_html :
priv->data->out_context_html;
		} else {
			/* content */
			html = consecutive ? priv->data->out_nextcontent_html :
priv->data->out_content_html;
		}

		/* remove all the unread marks when we are sending a message
*/
		theme_adium_remove_all_focus_marks (theme);
	} else {
		/* in */
		if (is_backlog) {
			/* context */
			html = consecutive ? priv->data->in_nextcontext_html :
priv->data->in_context_html;
```

```
            } else {
                  /* content */
                  html = consecutive ? priv->data->in_nextcontent_html :
priv->data->in_content_html;
            }
      }

      theme_adium_append_html (theme, func, html, body_escaped,
                        avatar_filename, name, contact_id,
                        service_name, message_classes->str,
                        timestamp, is_backlog, empathy_contact_is_user
(sender));

      /* Keep the sender of the last displayed message */
      if (priv->last_contact) {
            g_object_unref (priv->last_contact);
      }
      priv->last_contact = g_object_ref (sender);
      priv->last_timestamp = timestamp;
      priv->last_is_backlog = is_backlog;

      g_free (body_escaped);
      g_string_free (message_classes, TRUE);
}
<sep>
asmlinkage long sys_setrlimit(unsigned int resource, struct rlimit __user
*rlim)
{
      struct rlimit new_rlim, *old_rlim;
      unsigned long it_prof_secs;
      int retval;

      if (resource >= RLIM_NLIMITS)
            return -EINVAL;
      if (copy_from_user(&new_rlim, rlim, sizeof(*rlim)))
            return -EFAULT;
      if (new_rlim.rlim_cur > new_rlim.rlim_max)
            return -EINVAL;
      old_rlim = current->signal->rlim + resource;
      if ((new_rlim.rlim_max > old_rlim->rlim_max) &&
          !capable(CAP_SYS_RESOURCE))
            return -EPERM;
      if (resource == RLIMIT_NOFILE && new_rlim.rlim_max > NR_OPEN)
            return -EPERM;

      retval = security_task_setrlimit(resource, &new_rlim);
      if (retval)
            return retval;

      task_lock(current->group_leader);
      *old_rlim = new_rlim;
      task_unlock(current->group_leader);

      if (resource != RLIMIT_CPU)
```

```
            goto out;

      /*
       * RLIMIT_CPU handling.   Note that the kernel fails to return an
error
       * code if it rejected the user's attempt to set RLIMIT_CPU.  This
is a
       * very long-standing error, and fixing it now risks breakage of
       * applications, so we live with it
       */
      if (new_rlim.rlim_cur == RLIM_INFINITY)
            goto out;

      it_prof_secs = cputime_to_secs(current->signal->it_prof_expires);
      if (it_prof_secs == 0 || new_rlim.rlim_cur <= it_prof_secs) {
            unsigned long rlim_cur = new_rlim.rlim_cur;
            cputime_t cputime;

            if (rlim_cur == 0) {
                  /*
                   * The caller is asking for an immediate RLIMIT_CPU
                   * expiry.  But we use the zero value to mean "it was
                   * never set".  So let's cheat and make it one second
                   * instead
                   */
                  rlim_cur = 1;
            }
            cputime = secs_to_cputime(rlim_cur);
            read_lock(&tasklist_lock);
            spin_lock_irq(&current->sighand->siglock);
            set_process_cpu_timer(current, CPUCLOCK_PROF, &cputime,
NULL);
            spin_unlock_irq(&current->sighand->siglock);
            read_unlock(&tasklist_lock);
      }
out:
      return 0;
}
<sep>
static int check_chain_extensions(X509_STORE_CTX *ctx)
{
    int i, must_be_ca, plen = 0;
    X509 *x;
    int proxy_path_length = 0;
    int purpose;
    int allow_proxy_certs;
    int num = sk_X509_num(ctx->chain);

    /*-
     *  must_be_ca can have 1 of 3 values:
     * -1: we accept both CA and non-CA certificates, to allow direct
     *     use of self-signed certificates (which are marked as CA).
     * 0:  we only accept non-CA certificates.  This is currently not
     *     used, but the possibility is present for future extensions.
```

```
 * 1:  we only accept CA certificates.  This is currently used for
 *     all certificates in the chain except the leaf certificate.
 */
must_be_ca = -1;

/* CRL path validation */
if (ctx->parent) {
    allow_proxy_certs = 0;
    purpose = X509_PURPOSE_CRL_SIGN;
} else {
    allow_proxy_certs =
        ! !(ctx->param->flags & X509_V_FLAG_ALLOW_PROXY_CERTS);
    /*
     * A hack to keep people who don't want to modify their software
     * happy
     */
    if (getenv("OPENSSL_ALLOW_PROXY_CERTS"))
        allow_proxy_certs = 1;
    purpose = ctx->param->purpose;
}

for (i = 0; i < num; i++) {
    int ret;
    x = sk_X509_value(ctx->chain, i);
    if (!(ctx->param->flags & X509_V_FLAG_IGNORE_CRITICAL)
        && (x->ex_flags & EXFLAG_CRITICAL)) {
        ctx->error = X509_V_ERR_UNHANDLED_CRITICAL_EXTENSION;
        ctx->error_depth = i;
        ctx->current_cert = x;
        if (!ctx->verify_cb(0, ctx))
            return 0;
    }
    if (!allow_proxy_certs && (x->ex_flags & EXFLAG_PROXY)) {
        ctx->error = X509_V_ERR_PROXY_CERTIFICATES_NOT_ALLOWED;
        ctx->error_depth = i;
        ctx->current_cert = x;
        if (!ctx->verify_cb(0, ctx))
            return 0;
    }
    ret = X509_check_ca(x);
    switch (must_be_ca) {
    case -1:
        if ((ctx->param->flags & X509_V_FLAG_X509_STRICT)
            && (ret != 1) && (ret != 0)) {
            ret = 0;
            ctx->error = X509_V_ERR_INVALID_CA;
        } else
            ret = 1;
        break;
    case 0:
        if (ret != 0) {
            ret = 0;
            ctx->error = X509_V_ERR_INVALID_NON_CA;
        } else
```

```
                ret = 1;
            break;
        default:
            if ((ret == 0)
                || ((ctx->param->flags & X509_V_FLAG_X509_STRICT)
                    && (ret != 1))) {
                ret = 0;
                ctx->error = X509_V_ERR_INVALID_CA;
            } else
                ret = 1;
            break;
        }
        if (ret == 0) {
            ctx->error_depth = i;
            ctx->current_cert = x;
            if (! ctx->verify_cb(0, ctx))
                return 0;
        }
        if (purpose > 0) {
            if (!check_purpose(ctx, x, purpose, i, must_be_ca))
                return 0;
        }
        /* Check pathlen if not self issued */
        if ((i > 1) && !(x->ex_flags & EXFLAG_SI)
            && (x->ex_pathlen != -1)
            && (plen > (x->ex_pathlen + proxy_path_length + 1))) {
            ctx->error = X509_V_ERR_PATH_LENGTH_EXCEEDED;
            ctx->error_depth = i;
            ctx->current_cert = x;
            if (!ctx->verify_cb(0, ctx))
                return 0;
        }
        /* Increment path length if not self issued */
        if (!(x->ex_flags & EXFLAG_SI))
            plen++;
        /*
         * If this certificate is a proxy certificate, the next
certificate
         * must be another proxy certificate or a EE certificate.  If
not,
         * the next certificate must be a CA certificate.
         */
        if (x->ex_flags & EXFLAG_PROXY) {
            if (x->ex_pcpathlen != -1 && i > x->ex_pcpathlen) {
                ctx->error = X509_V_ERR_PROXY_PATH_LENGTH_EXCEEDED;
                ctx->error_depth = i;
                ctx->current_cert = x;
                if (!ctx->verify_cb(0, ctx))
                    return 0;
            }
            proxy_path_length++;
            must_be_ca = 0;
        } else
            must_be_ca = 1;
```

```
    }
    return 1;
}
<sep>
_dopr(char **sbuffer,
      char **buffer,
      size_t *maxlen,
      size_t *retlen, int *truncated, const char *format, va_list args)
{
    char ch;
    LLONG value;
    LDOUBLE fvalue;
    char *strvalue;
    int min;
    int max;
    int state;
    int flags;
    int cflags;
    size_t currlen;

    state = DP_S_DEFAULT;
    flags = currlen = cflags = min = 0;
    max = -1;
    ch = *format++;

    while (state != DP_S_DONE) {
        if (ch == '\0' || (buffer == NULL && currlen >= *maxlen))
            state = DP_S_DONE;

        switch (state) {
        case DP_S_DEFAULT:
            if (ch == '%')
                state = DP_S_FLAGS;
            else
                doapr_outch(sbuffer, buffer, &currlen, maxlen, ch);
            ch = *format++;
            break;
        case DP_S_FLAGS:
            switch (ch) {
            case '-':
                flags |= DP_F_MINUS;
                ch = *format++;
                break;
            case '+':
                flags |= DP_F_PLUS;
                ch = *format++;
                break;
            case ' ':
                flags |= DP_F_SPACE;
                ch = *format++;
                break;
            case '#':
                flags |= DP_F_NUM;
                ch = *format++;
```

```
            break;
        case '0':
            flags |= DP_F_ZERO;
            ch = *format++;
            break;
        default:
            state = DP_S_MIN;
            break;
        }
        break;
    case DP_S_MIN:
        if (isdigit((unsigned char)ch)) {
            min = 10 * min + char_to_int(ch);
            ch = *format++;
        } else if (ch == '*') {
            min = va_arg(args, int);
            ch = *format++;
            state = DP_S_DOT;
        } else
            state = DP_S_DOT;
        break;
    case DP_S_DOT:
        if (ch == '.') {
            state = DP_S_MAX;
            ch = *format++;
        } else
            state = DP_S_MOD;
        break;
    case DP_S_MAX:
        if (isdigit((unsigned char)ch)) {
            if (max < 0)
                max = 0;
            max = 10 * max + char_to_int(ch);
            ch = *format++;
        } else if (ch == '*') {
            max = va_arg(args, int);
            ch = *format++;
            state = DP_S_MOD;
        } else
            state = DP_S_MOD;
        break;
    case DP_S_MOD:
        switch (ch) {
        case 'h':
            cflags = DP_C_SHORT;
            ch = *format++;
            break;
        case 'l':
            if (*format == 'l') {
                cflags = DP_C_LLONG;
                format++;
            } else
                cflags = DP_C_LONG;
            ch = *format++;
```

```
                break;
            case 'q':
                cflags = DP_C_LLONG;
                ch = *format++;
                break;
            case 'L':
                cflags = DP_C_LDOUBLE;
                ch = *format++;
                break;
            default:
                break;
            }
            state = DP_S_CONV;
            break;
        case DP_S_CONV:
            switch (ch) {
            case 'd':
            case 'i':
                switch (cflags) {
                case DP_C_SHORT:
                    value = (short int)va_arg(args, int);
                    break;
                case DP_C_LONG:
                    value = va_arg(args, long int);
                    break;
                case DP_C_LLONG:
                    value = va_arg(args, LLONG);
                    break;
                default:
                    value = va_arg(args, int);
                    break;
                }
                fmtint(sbuffer, buffer, &currlen, maxlen,
                       value, 10, min, max, flags);
                break;
            case 'X':
                flags |= DP_F_UP;
                /* FALLTHROUGH */
            case 'x':
            case 'o':
            case 'u':
                flags |= DP_F_UNSIGNED;
                switch (cflags) {
                case DP_C_SHORT:
                    value = (unsigned short int)va_arg(args, unsigned
int);
                    break;
                case DP_C_LONG:
                    value = (LLONG) va_arg(args, unsigned long int);
                    break;
                case DP_C_LLONG:
                    value = va_arg(args, unsigned LLONG);
                    break;
                default:
```

```
            value = (LLONG) va_arg(args, unsigned int);
            break;
        }
        fmtint(sbuffer, buffer, &currlen, maxlen, value,
               ch == 'o' ? 8 : (ch == 'u' ? 10 : 16),
               min, max, flags);
        break;
    case 'f':
        if (cflags == DP_C_LDOUBLE)
            fvalue = va_arg(args, LDOUBLE);
        else
            fvalue = va_arg(args, double);
        fmtfp(sbuffer, buffer, &currlen, maxlen,
              fvalue, min, max, flags);
        break;
    case 'E':
        flags |= DP_F_UP;
    case 'e':
        if (cflags == DP_C_LDOUBLE)
            fvalue = va_arg(args, LDOUBLE);
        else
            fvalue = va_arg(args, double);
        break;
    case 'G':
        flags |= DP_F_UP;
    case 'g':
        if (cflags == DP_C_LDOUBLE)
            fvalue = va_arg(args, LDOUBLE);
        else
            fvalue = va_arg(args, double);
        break;
    case 'c':
        doapr_outch(sbuffer, buffer, &currlen, maxlen,
                    va_arg(args, int));
        break;
    case 's':
        strvalue = va_arg(args, char *);
        if (max < 0) {
            if (buffer)
                max = INT_MAX;
            else
                max = *maxlen;
        }
        fmtstr(sbuffer, buffer, &currlen, maxlen, strvalue,
               flags, min, max);
        break;
    case 'p':
        value = (long)va_arg(args, void *);
        fmtint(sbuffer, buffer, &currlen, maxlen,
               value, 16, min, max, flags | DP_F_NUM);
        break;
    case 'n':          /* XXX */
        if (cflags == DP_C_SHORT) {
            short int *num;
```

```
                    num = va_arg(args, short int *);
                    *num = currlen;
                } else if (cflags == DP_C_LONG) { /* XXX */
                    long int *num;
                    num = va_arg(args, long int *);
                    *num = (long int)currlen;
                } else if (cflags == DP_C_LLONG) { /* XXX */
                    LLONG *num;
                    num = va_arg(args, LLONG *);
                    *num = (LLONG) currlen;
                } else {
                    int *num;
                    num = va_arg(args, int *);
                    *num = currlen;
                }
                break;
            case '%':
                doapr_outch(sbuffer, buffer, &currlen, maxlen, ch);
                break;
            case 'w':
                /* not supported yet, treat as next char */
                ch = *format++;
                break;
            default:
                /* unknown, skip */
                break;
            }
            ch = *format++;
            state = DP_S_DEFAULT;
            flags = cflags = min = 0;
            max = -1;
            break;
        case DP_S_DONE:
            break;
        default:
            break;
        }
    }
    *truncated = (currlen > *maxlen - 1);
    if (*truncated)
        currlen = *maxlen - 1;
    doapr_outch(sbuffer, buffer, &currlen, maxlen, '\0');
    *retlen = currlen - 1;
    return;
}
<sep>
unquoted_glob_pattern_p (string)
     register char *string;
{
  register int c;
  char *send;
  int open, bsquote;

  DECLARE_MBSTATE;
```

```
  open = bsquote = 0;
  send = string + strlen (string);

  while (c = *string++)
    {
      switch (c)
      {
      case '?':
      case '*':
        return (1);

      case '[':
        open++;
        continue;

      case ']':
        if (open)
          return (1);
        continue;

      case '+':
      case '@':
      case '!':
        if (*string == '(')  /*)*/
          return (1);
        continue;

      /* A pattern can't end with a backslash, but a backslash in the
pattern
         can be removed by the matching engine, so we have to run it
through
         globbing. */
      case '\\':
        if (*string != '\0' && *string != '/')
          {
            bsquote = 1;
            string++;
            continue;
          }
        else if (*string == 0)
          return (0);

      case CTLESC:
        if (*string++ == '\0')
          return (0);
      }

      /* Advance one fewer byte than an entire multibyte character to
       account for the auto-increment in the loop above. */
#ifdef HANDLE_MULTIBYTE
      string--;
      ADVANCE_CHAR_P (string, send - string);
      string++;
```

```
#else
      ADVANCE_CHAR_P (string, send - string);
#endif
    }

  return ((bsquote && posix_glob_backslash) ? 2 : 0);
}
<sep>
void vbe_ioport_write_data(void *opaque, uint32_t addr, uint32_t val)
{
    VGACommonState *s = opaque;

    if (s->vbe_index <= VBE_DISPI_INDEX_NB) {
#ifdef DEBUG_BOCHS_VBE
        printf("VBE: write index=0x%x val=0x%x\n", s->vbe_index, val);
#endif
        switch(s->vbe_index) {
        case VBE_DISPI_INDEX_ID:
            if (val == VBE_DISPI_ID0 ||
                val == VBE_DISPI_ID1 ||
                val == VBE_DISPI_ID2 ||
                val == VBE_DISPI_ID3 ||
                val == VBE_DISPI_ID4) {
                s->vbe_regs[s->vbe_index] = val;
            }
            break;
        case VBE_DISPI_INDEX_XRES:
            if ((val <= VBE_DISPI_MAX_XRES) && ((val & 7) == 0)) {
                s->vbe_regs[s->vbe_index] = val;
            }
            break;
        case VBE_DISPI_INDEX_YRES:
            if (val <= VBE_DISPI_MAX_YRES) {
                s->vbe_regs[s->vbe_index] = val;
            }
            break;
        case VBE_DISPI_INDEX_BPP:
            if (val == 0)
                val = 8;
            if (val == 4 || val == 8 || val == 15 ||
                val == 16 || val == 24 || val == 32) {
                s->vbe_regs[s->vbe_index] = val;
            }
            break;
        case VBE_DISPI_INDEX_BANK:
            if (s->vbe_regs[VBE_DISPI_INDEX_BPP] == 4) {
              val &= (s->vbe_bank_mask >> 2);
            } else {
              val &= s->vbe_bank_mask;
            }
            s->vbe_regs[s->vbe_index] = val;
            s->bank_offset = (val << 16);
            vga_update_memory_access(s);
            break;
```

```
        case VBE_DISPI_INDEX_ENABLE:
            if ((val & VBE_DISPI_ENABLED) &&
                !(s->vbe_regs[VBE_DISPI_INDEX_ENABLE] &
VBE_DISPI_ENABLED)) {
                int h, shift_control;

                s->vbe_regs[VBE_DISPI_INDEX_VIRT_WIDTH] =
                    s->vbe_regs[VBE_DISPI_INDEX_XRES];
                s->vbe_regs[VBE_DISPI_INDEX_VIRT_HEIGHT] =
                    s->vbe_regs[VBE_DISPI_INDEX_YRES];
                s->vbe_regs[VBE_DISPI_INDEX_X_OFFSET] = 0;
                s->vbe_regs[VBE_DISPI_INDEX_Y_OFFSET] = 0;

                if (s->vbe_regs[VBE_DISPI_INDEX_BPP] == 4)
                    s->vbe_line_offset = s-
>vbe_regs[VBE_DISPI_INDEX_XRES] >> 1;
                else
                    s->vbe_line_offset = s-
>vbe_regs[VBE_DISPI_INDEX_XRES] *
                        ((s->vbe_regs[VBE_DISPI_INDEX_BPP] + 7) >> 3);
                s->vbe_start_addr = 0;

                /* clear the screen (should be done in BIOS) */
                if (!(val & VBE_DISPI_NOCLEARMEM)) {
                    memset(s->vram_ptr, 0,
                           s->vbe_regs[VBE_DISPI_INDEX_YRES] * s-
>vbe_line_offset);
                }

                /* we initialize the VGA graphic mode (should be done
                   in BIOS) */
                /* graphic mode + memory map 1 */
                s->gr[VGA_GFX_MISC] = (s->gr[VGA_GFX_MISC] & ~0x0c) |
0x04 |
                    VGA_GR06_GRAPHICS_MODE;
                s->cr[VGA_CRTC_MODE] |= 3; /* no CGA modes */
                s->cr[VGA_CRTC_OFFSET] = s->vbe_line_offset >> 3;
                /* width */
                s->cr[VGA_CRTC_H_DISP] =
                    (s->vbe_regs[VBE_DISPI_INDEX_XRES] >> 3) - 1;
                /* height (only meaningful if < 1024) */
                h = s->vbe_regs[VBE_DISPI_INDEX_YRES] - 1;
                s->cr[VGA_CRTC_V_DISP_END] = h;
                s->cr[VGA_CRTC_OVERFLOW] = (s->cr[VGA_CRTC_OVERFLOW] &
~0x42) |
                    ((h >> 7) & 0x02) | ((h >> 3) & 0x40);
                /* line compare to 1023 */
                s->cr[VGA_CRTC_LINE_COMPARE] = 0xff;
                s->cr[VGA_CRTC_OVERFLOW] |= 0x10;
                s->cr[VGA_CRTC_MAX_SCAN] |= 0x40;

                if (s->vbe_regs[VBE_DISPI_INDEX_BPP] == 4) {
                    shift_control = 0;
                    s->sr[VGA_SEQ_CLOCK_MODE] &= ~8; /* no double line */
```

```
                } else {
                    shift_control = 2;
                    /* set chain 4 mode */
                    s->sr[VGA_SEQ_MEMORY_MODE] |= VGA_SR04_CHN_4M;
                    /* activate all planes */
                    s->sr[VGA_SEQ_PLANE_WRITE] |= VGA_SR02_ALL_PLANES;
                }
                s->gr[VGA_GFX_MODE] = (s->gr[VGA_GFX_MODE] & ~0x60) |
                    (shift_control << 5);
                s->cr[VGA_CRTC_MAX_SCAN] &= ~0x9f; /* no double scan */
            } else {
                /* XXX: the bios should do that */
                s->bank_offset = 0;
            }
            s->dac_8bit = (val & VBE_DISPI_8BIT_DAC) > 0;
            s->vbe_regs[s->vbe_index] = val;
            vga_update_memory_access(s);
            break;
        case VBE_DISPI_INDEX_VIRT_WIDTH:
            {
                int w, h, line_offset;

                if (val < s->vbe_regs[VBE_DISPI_INDEX_XRES])
                    return;
                w = val;
                if (s->vbe_regs[VBE_DISPI_INDEX_BPP] == 4)
                    line_offset = w >> 1;
                else
                    line_offset = w * ((s->vbe_regs[VBE_DISPI_INDEX_BPP]
+ 7) >> 3);
                h = s->vbe_size / line_offset;
                /* XXX: support weird bochs semantics ? */
                if (h < s->vbe_regs[VBE_DISPI_INDEX_YRES])
                    return;
                s->vbe_regs[VBE_DISPI_INDEX_VIRT_WIDTH] = w;
                s->vbe_regs[VBE_DISPI_INDEX_VIRT_HEIGHT] = h;
                s->vbe_line_offset = line_offset;
            }
            break;
        case VBE_DISPI_INDEX_X_OFFSET:
        case VBE_DISPI_INDEX_Y_OFFSET:
            {
                int x;
                s->vbe_regs[s->vbe_index] = val;
                s->vbe_start_addr = s->vbe_line_offset * s-
>vbe_regs[VBE_DISPI_INDEX_Y_OFFSET];
                x = s->vbe_regs[VBE_DISPI_INDEX_X_OFFSET];
                if (s->vbe_regs[VBE_DISPI_INDEX_BPP] == 4)
                    s->vbe_start_addr += x >> 1;
                else
                    s->vbe_start_addr += x * ((s-
>vbe_regs[VBE_DISPI_INDEX_BPP] + 7) >> 3);
                s->vbe_start_addr >>= 2;
            }
```

```
            break;
        default:
            break;
        }
    }
}
<sep>
void SSL_SESSION_free(SSL_SESSION *ss)
      {
      int i;

      if(ss == NULL)
          return;

      i=CRYPTO_add(&ss->references,-1,CRYPTO_LOCK_SSL_SESSION);
#ifdef REF_PRINT
      REF_PRINT("SSL_SESSION",ss);
#endif
      if (i > 0) return;
#ifdef REF_CHECK
      if (i < 0)
            {
            fprintf(stderr,"SSL_SESSION_free, bad reference count\n");
            abort(); /* ok */
            }
#endif

      CRYPTO_free_ex_data(CRYPTO_EX_INDEX_SSL_SESSION, ss, &ss->ex_data);

      OPENSSL_cleanse(ss->key_arg,sizeof ss->key_arg);
      OPENSSL_cleanse(ss->master_key,sizeof ss->master_key);
      OPENSSL_cleanse(ss->session_id,sizeof ss->session_id);
      if (ss->sess_cert != NULL) ssl_sess_cert_free(ss->sess_cert);
      if (ss->peer != NULL) X509_free(ss->peer);
      if (ss->ciphers != NULL) sk_SSL_CIPHER_free(ss->ciphers);
#ifndef OPENSSL_NO_TLSEXT
      if (ss->tlsext_hostname != NULL) OPENSSL_free(ss->tlsext_hostname);
#endif
#ifndef OPENSSL_NO_PSK
      if (ss->psk_identity_hint != NULL)
            OPENSSL_free(ss->psk_identity_hint);
      if (ss->psk_identity != NULL)
            OPENSSL_free(ss->psk_identity);
#endif
      OPENSSL_cleanse(ss,sizeof(*ss));
      OPENSSL_free(ss);
      }
<sep>
static void ssdp_recv(int sd)
{
      ssize_t len;
      struct sockaddr sa;
      socklen_t salen;
      char buf[MAX_PKT_SIZE];
```

```
      memset(buf, 0, sizeof(buf));
      len = recvfrom(sd, buf, sizeof(buf), MSG_DONTWAIT, &sa, &salen);
      if (len > 0) {
            buf[len] = 0;

            if (sa.sa_family != AF_INET)
                  return;

            if (strstr(buf, "M-SEARCH *")) {
                  size_t i;
                  char *ptr, *type;
                  struct ifsock *ifs;
                  struct sockaddr_in *sin = (struct sockaddr_in *)&sa;

                  ifs = find_outbound(&sa);
                  if (!ifs) {
                        logit(LOG_DEBUG, "No matching socket for client
%s", inet_ntoa(sin->sin_addr));
                        return;
                  }
                  logit(LOG_DEBUG, "Matching socket for client %s",
inet_ntoa(sin->sin_addr));

                  type = strcasestr(buf, "\r\nST:");
                  if (!type) {
                        logit(LOG_DEBUG, "No Search Type (ST:) found in M-
SEARCH *, assuming " SSDP_ST_ALL);
                        type = SSDP_ST_ALL;
                        send_message(ifs, type, &sa);
                        return;
                  }

                  type = strchr(type, ':');
                  if (!type)
                        return;
                  type++;
                  while (isspace(*type))
                        type++;

                  ptr = strstr(type, "\r\n");
                  if (!ptr)
                        return;
                  *ptr = 0;

                  for (i = 0; supported_types[i]; i++) {
                        if (!strcmp(supported_types[i], type)) {
                              logit(LOG_DEBUG, "M-SEARCH * ST: %s from %s
port %d", type,
                                    inet_ntoa(sin->sin_addr), ntohs(sin-
>sin_port));
                              send_message(ifs, type, &sa);
                              return;
                        }
```

```
                }

                logit(LOG_DEBUG, "M-SEARCH * for unsupported ST: %s from
%s", type,
                      inet_ntoa(sin->sin_addr));
            }
      }
}
<sep>
int mongo_env_read_socket( mongo *conn, void *buf, int len ) {
    char *cbuf = buf;
    while ( len ) {
        int sent = recv( conn->sock, cbuf, len, 0 );
        if ( sent == 0 || sent == -1 ) {
            conn->err = MONGO_IO_ERROR;
            return MONGO_ERROR;
        }
        cbuf += sent;
        len -= sent;
    }

    return MONGO_OK;
}
<sep>
zsethalftone5(i_ctx_t *i_ctx_p)
{
    os_ptr op = osp;
    uint count;
    gs_halftone_component *phtc = 0;
    gs_halftone_component *pc;
    int code = 0;
    int j;
    bool have_default;
    gs_halftone *pht = 0;
    gx_device_halftone *pdht = 0;
    ref sprocs[GS_CLIENT_COLOR_MAX_COMPONENTS + 1];
    ref tprocs[GS_CLIENT_COLOR_MAX_COMPONENTS + 1];
    gs_memory_t *mem;
    uint edepth = ref_stack_count(&e_stack);
    int npop = 2;
    int dict_enum = dict_first(op);
    ref rvalue[2];
    int cname, colorant_number;
    byte * pname;
    uint name_size;
    int halftonetype, type = 0;
    gs_gstate *pgs = igs;
    int space_index = r_space_index(op - 1);

    mem = (gs_memory_t *) idmemory->spaces_indexed[space_index];

    check_type(*op, t_dictionary);
    check_dict_read(*op);
    check_type(op[-1], t_dictionary);
```

```
    check_dict_read(op[-1]);

    /*
     * We think that Type 2 and Type 4 halftones, like
     * screens set by setcolorscreen, adapt automatically to
     * the device color space, so we need to mark them
     * with a different internal halftone type.
     */
    code = dict_int_param(op - 1, "HalftoneType", 1, 100, 0, &type);
    if (code < 0)
          return code;
    halftonetype = (type == 2 || type == 4)
                        ? ht_type_multiple_colorscreen
                        : ht_type_multiple;

    /* Count how many components that we will actually use. */

    have_default = false;
    for (count = 0; ;) {

        /* Move to next element in the dictionary */
        if ((dict_enum = dict_next(op, dict_enum, rvalue)) == -1)
            break;
        /*
         * Verify that we have a valid component.  We may have a
         * /HalfToneType entry.
         */
        if (!r_has_type(&rvalue[0], t_name))
            continue;
        if (!r_has_type(&rvalue[1], t_dictionary))
            continue;

        /* Get the name of the component  verify that we will use it. */
        cname = name_index(mem, &rvalue[0]);
        code = gs_get_colorname_string(mem, cname, &pname, &name_size);
        if (code < 0)
            break;
        colorant_number = gs_cname_to_colorant_number(pgs, pname,
name_size,
                                                      halftonetype);
        if (colorant_number < 0)
            continue;
        else if (colorant_number == GX_DEVICE_COLOR_MAX_COMPONENTS) {
            /* If here then we have the "Default" component */
            if (have_default)
                return_error(gs_error_rangecheck);
            have_default = true;
        }

        count++;
        /*
         * Check to see if we have already reached the legal number of
         * components.
         */
```

```
        if (count > GS_CLIENT_COLOR_MAX_COMPONENTS + 1) {
            code = gs_note_error(gs_error_rangecheck);
            break;
        }
    }
    if (count == 0 || (halftonetype == ht_type_multiple && !
have_default))
        code = gs_note_error(gs_error_rangecheck);

    if (code >= 0) {
        check_estack(5);           /* for sampling Type 1 screens */
        refset_null(sprocs, count);
        refset_null(tprocs, count);
        rc_alloc_struct_0(pht, gs_halftone, &st_halftone,
                          imemory, pht = 0, ".sethalftone5");
        phtc = gs_alloc_struct_array(mem, count, gs_halftone_component,
                                     &st_ht_component_element,
                                     ".sethalftone5");
        rc_alloc_struct_0(pdht, gx_device_halftone, &st_device_halftone,
                          imemory, pdht = 0, ".sethalftone5");
        if (pht == 0 || phtc == 0 || pdht == 0) {
            j = 0; /* Quiet the compiler:
                      gs_note_error isn't necessarily identity,
                      so j could be left ununitialized. */
            code = gs_note_error(gs_error_VMerror);
        }
    }
    if (code >= 0) {
        dict_enum = dict_first(op);
        for (j = 0, pc = phtc; ;) {
            int type;

            /* Move to next element in the dictionary */
            if ((dict_enum = dict_next(op, dict_enum, rvalue)) == -1)
                break;
            /*
             * Verify that we have a valid component.  We may have a
             * /HalfToneType entry.
             */
            if (!r_has_type(&rvalue[0], t_name))
                continue;
            if (!r_has_type(&rvalue[1], t_dictionary))
                continue;

            /* Get the name of the component */
            cname = name_index(mem, &rvalue[0]);
            code = gs_get_colorname_string(mem, cname, &pname,
&name_size);
            if (code < 0)
                break;
            colorant_number = gs_cname_to_colorant_number(pgs, pname,
name_size,
                                                          halftonetype);
            if (colorant_number < 0)
```

```
                continue;          /* Do not use this component */
            pc->cname = cname;
            pc->comp_number = colorant_number;

            /* Now process the component dictionary */
            check_dict_read(rvalue[1]);
            if (dict_int_param(&rvalue[1], "HalftoneType", 1, 7, 0,
&type) < 0) {
                code = gs_note_error(gs_error_typecheck);
                break;
            }
            switch (type) {
                default:
                    code = gs_note_error(gs_error_rangecheck);
                    break;
                case 1:
                    code = dict_spot_params(&rvalue[1], &pc->params.spot,
                                            sprocs + j, tprocs + j,
mem);
                    pc->params.spot.screen.spot_function = spot1_dummy;
                    pc->type = ht_type_spot;
                    break;
                case 3:
                    code = dict_threshold_params(&rvalue[1], &pc-
>params.threshold,
                                                 tprocs + j);
                    pc->type = ht_type_threshold;
                    break;
                case 7:
                    code = dict_threshold2_params(&rvalue[1], &pc-
>params.threshold2,
                                                  tprocs + j,
imemory);
                    pc->type = ht_type_threshold2;
                    break;
            }
            if (code < 0)
                break;
            pc++;
            j++;
        }
    }
    if (code >= 0) {
        pht->type = halftonetype;
        pht->params.multiple.components = phtc;
        pht->params.multiple.num_comp = j;
        pht->params.multiple.get_colorname_string =
gs_get_colorname_string;
        code = gs_sethalftone_prepare(igs, pht, pdht);
    }
    if (code >= 0) {
        /*
         * Put the actual frequency and angle in the spot function
component dictionaries.
```

```
         */
        dict_enum = dict_first(op);
        for (pc = phtc; ; ) {
            /* Move to next element in the dictionary */
            if ((dict_enum = dict_next(op, dict_enum, rvalue)) == -1)
                break;

            /* Verify that we have a valid component */
            if (!r_has_type(&rvalue[0], t_name))
                continue;
            if (!r_has_type(&rvalue[1], t_dictionary))
                continue;

            /* Get the name of the component and verify that we will use
it. */
            cname = name_index(mem, &rvalue[0]);
            code = gs_get_colorname_string(mem, cname, &pname,
&name_size);
            if (code < 0)
                break;
            colorant_number = gs_cname_to_colorant_number(pgs, pname,
name_size,
                                                          halftonetype);
            if (colorant_number < 0)
                continue;

            if (pc->type == ht_type_spot) {
                code = dict_spot_results(i_ctx_p, &rvalue[1], &pc-
>params.spot);
                if (code < 0)
                    break;
            }
            pc++;
        }
    }
    if (code >= 0) {
        /*
         * Schedule the sampling of any Type 1 screens,
         * and any (Type 1 or Type 3) TransferFunctions.
         * Save the stack depths in case we have to back out.
         */
        uint odepth = ref_stack_count(&o_stack);
        ref odict, odict5;

        odict = op[-1];
        odict5 = *op;
        pop(2);
        op = osp;
        esp += 5;
        make_mark_estack(esp - 4, es_other, sethalftone_cleanup);
        esp[-3] = odict;
        make_istruct(esp - 2, 0, pht);
        make_istruct(esp - 1, 0, pdht);
        make_op_estack(esp, sethalftone_finish);
```

```
        for (j = 0; j < count; j++) {
            gx_ht_order *porder = NULL;

            if (pdht->components == 0)
                porder = &pdht->order;
            else {
                /* Find the component in pdht that matches component j in
                   the pht; gs_sethalftone_prepare() may permute these.
*/
                int k;
                int comp_number = phtc[j].comp_number;
                for (k = 0; k < count; k++) {
                    if (pdht->components[k].comp_number == comp_number) {
                        porder = &pdht->components[k].corder;
                        break;
                    }
                }
            }
            switch (phtc[j].type) {
            case ht_type_spot:
                code = zscreen_enum_init(i_ctx_p, porder,
                                         &phtc[j].params.spot.screen,
                                         &sprocs[j], 0, 0, space_index);
                if (code < 0)
                    break;
                /* falls through */
            case ht_type_threshold:
                if (!r_has_type(tprocs + j, t__invalid)) {
                    /* Schedule TransferFunction sampling. */
                    /****** check_xstack IS WRONG ******/
                    check_ostack(zcolor_remap_one_ostack);
                    check_estack(zcolor_remap_one_estack);
                    code = zcolor_remap_one(i_ctx_p, tprocs + j,
                                            porder->transfer, igs,
                                            zcolor_remap_one_finish);
                    op = osp;
                }
                break;
            default:   /* not possible here, but to keep */
                                /* the compilers happy.... */
                ;
            }
            if (code < 0) {  /* Restore the stack. */
                ref_stack_pop_to(&o_stack, odepth);
                ref_stack_pop_to(&e_stack, edepth);
                op = osp;
                op[-1] = odict;
                *op = odict5;
                break;
            }
            npop = 0;
        }
    }
    if (code < 0) {
```

```
        gs_free_object(mem, pdht, ".sethalftone5");
        gs_free_object(mem, phtc, ".sethalftone5");
        gs_free_object(mem, pht, ".sethalftone5");
        return code;
    }
    pop(npop);
    return (ref_stack_count(&e_stack) > edepth ? o_push_estack : 0);
}
<sep>
am_cache_entry_t *am_new_request_session(request_rec *r)
{
    const char *session_id;

    /* Generate session id. */
    session_id = am_generate_id(r);
    if(session_id == NULL) {
        ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r,
                      "Error creating session id.");
        return NULL;
    }


    /* Set session id. */
    am_cookie_set(r, session_id);

    return am_cache_new(r->server, session_id);
}
<sep>
static void ati_cursor_draw_line(VGACommonState *vga, uint8_t *d, int
scr_y)
{
    ATIVGAState *s = container_of(vga, ATIVGAState, vga);
    uint8_t *src;
    uint32_t *dp = (uint32_t *)d;
    int i, j, h;

    if (!(s->regs.crtc_gen_cntl & CRTC2_CUR_EN) ||
        scr_y < vga->hw_cursor_y || scr_y >= vga->hw_cursor_y + 64 ||
        scr_y > s->regs.crtc_v_total_disp >> 16) {
        return;
    }
    /* FIXME handle cur_hv_offs correctly */
    src = s->vga.vram_ptr + s->cursor_offset + (scr_y - vga->hw_cursor_y)
* 16;
    dp = &dp[vga->hw_cursor_x];
    h = ((s->regs.crtc_h_total_disp >> 16) + 1) * 8;
    for (i = 0; i < 8; i++) {
        uint32_t color;
        uint8_t abits = src[i];
        uint8_t xbits = src[i + 8];
        for (j = 0; j < 8; j++, abits <<= 1, xbits <<= 1) {
            if (abits & BIT(7)) {
                if (xbits & BIT(7)) {
                    color = dp[i * 8 + j] ^ 0xffffffff; /* complement */
```

```
                } else {
                    continue; /* transparent, no change */
                }
            } else {
                color = (xbits & BIT(7) ? s->regs.cur_color1 :
                                          s->regs.cur_color0) |
0xff000000;
            }
            if (vga->hw_cursor_x + i * 8 + j >= h) {
                return; /* end of screen, don't span to next line */
            }
            dp[i * 8 + j] = color;
        }
    }
}
<sep>
static int tftp_session_allocate(Slirp *slirp, struct sockaddr_storage
*srcsas,
                                 struct tftp_t *tp)
{
    struct tftp_session *spt;
    int k;

    for (k = 0; k < TFTP_SESSIONS_MAX; k++) {
        spt = &slirp->tftp_sessions[k];

        if (!tftp_session_in_use(spt))
            goto found;

        /* sessions time out after 5 inactive seconds */
        if ((int)(curtime - spt->timestamp) > 5000) {
            tftp_session_terminate(spt);
            goto found;
        }
    }

    return -1;

found:
    memset(spt, 0, sizeof(*spt));
    memcpy(&spt->client_addr, srcsas, sockaddr_size(srcsas));
    spt->fd = -1;
    spt->block_size = 512;
    spt->client_port = tp->udp.uh_sport;
    spt->slirp = slirp;

    tftp_session_update(spt);

    return k;
}
<sep>
int ssl3_get_client_key_exchange(SSL *s)
{
    int i, al, ok;
```

```
    long n;
    unsigned long alg_k;
    unsigned char *p;
#ifndef OPENSSL_NO_RSA
    RSA *rsa = NULL;
    EVP_PKEY *pkey = NULL;
#endif
#ifndef OPENSSL_NO_DH
    BIGNUM *pub = NULL;
    DH *dh_srvr, *dh_clnt = NULL;
#endif
#ifndef OPENSSL_NO_KRB5
    KSSL_ERR kssl_err;
#endif                          /* OPENSSL_NO_KRB5 */

#ifndef OPENSSL_NO_ECDH
    EC_KEY *srvr_ecdh = NULL;
    EVP_PKEY *clnt_pub_pkey = NULL;
    EC_POINT *clnt_ecpoint = NULL;
    BN_CTX *bn_ctx = NULL;
#endif

    n = s->method->ssl_get_message(s,
                                   SSL3_ST_SR_KEY_EXCH_A,
                                   SSL3_ST_SR_KEY_EXCH_B,
                                   SSL3_MT_CLIENT_KEY_EXCHANGE, 2048,
&ok);

    if (!ok)
        return ((int)n);
    p = (unsigned char *)s->init_msg;

    alg_k = s->s3->tmp.new_cipher->algorithm_mkey;

#ifndef OPENSSL_NO_RSA
    if (alg_k & SSL_kRSA) {
        unsigned char rand_premaster_secret[SSL_MAX_MASTER_KEY_LENGTH];
        int decrypt_len;
        unsigned char decrypt_good, version_good;
        size_t j;

        /* FIX THIS UP EAY EAY EAY EAY */
        if (s->s3->tmp.use_rsa_tmp) {
            if ((s->cert != NULL) && (s->cert->rsa_tmp != NULL))
                rsa = s->cert->rsa_tmp;
            /*
             * Don't do a callback because rsa_tmp should be sent already
             */
            if (rsa == NULL) {
                al = SSL_AD_HANDSHAKE_FAILURE;
                SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                       SSL_R_MISSING_TMP_RSA_PKEY);
                goto f_err;
```

```
            }
        } else {
            pkey = s->cert->pkeys[SSL_PKEY_RSA_ENC].privatekey;
            if ((pkey == NULL) ||
                (pkey->type != EVP_PKEY_RSA) || (pkey->pkey.rsa == NULL))
{
                al = SSL_AD_HANDSHAKE_FAILURE;
                SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                       SSL_R_MISSING_RSA_CERTIFICATE);
                goto f_err;
            }
            rsa = pkey->pkey.rsa;
        }

        /* TLS and [incidentally] DTLS{0xFEFF} */
        if (s->version > SSL3_VERSION && s->version != DTLS1_BAD_VER) {
            n2s(p, i);
            if (n != i + 2) {
                if (!(s->options & SSL_OP_TLS_D5_BUG)) {
                    al = SSL_AD_DECODE_ERROR;
                    SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,

SSL_R_TLS_RSA_ENCRYPTED_VALUE_LENGTH_IS_WRONG);
                    goto f_err;
                } else
                    p -= 2;
            } else
                n = i;
        }

        /*
         * Reject overly short RSA ciphertext because we want to be sure
         * that the buffer size makes it safe to iterate over the entire
         * size of a premaster secret (SSL_MAX_MASTER_KEY_LENGTH). The
         * actual expected size is larger due to RSA padding, but the
         * bound is sufficient to be safe.
         */
        if (n < SSL_MAX_MASTER_KEY_LENGTH) {
            al = SSL_AD_DECRYPT_ERROR;
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_TLS_RSA_ENCRYPTED_VALUE_LENGTH_IS_WRONG);
            goto f_err;
        }

        /*
         * We must not leak whether a decryption failure occurs because
of
         * Bleichenbacher's attack on PKCS #1 v1.5 RSA padding (see RFC
2246,
         * section 7.4.7.1). The code follows that advice of the TLS RFC
and
         * generates a random premaster secret for the case that the
decrypt
         * fails. See https://tools.ietf.org/html/rfc5246#section-7.4.7.1
```

```
         */

        /*
         * should be RAND_bytes, but we cannot work around a failure.
         */
        if (RAND_pseudo_bytes(rand_premaster_secret,
                              sizeof(rand_premaster_secret)) <= 0)
            goto err;
        decrypt_len =
            RSA_private_decrypt((int)n, p, p, rsa, RSA_PKCS1_PADDING);
        ERR_clear_error();

        /*
         * decrypt_len should be SSL_MAX_MASTER_KEY_LENGTH. decrypt_good
will
         * be 0xff if so and zero otherwise.
         */
        decrypt_good =
            constant_time_eq_int_8(decrypt_len,
SSL_MAX_MASTER_KEY_LENGTH);

        /*
         * If the version in the decrypted pre-master secret is correct
then
         * version_good will be 0xff, otherwise it'll be zero. The
         * Klima-Pokorny-Rosa extension of Bleichenbacher's attack
         * (http://eprint.iacr.org/2003/052/) exploits the version number
         * check as a "bad version oracle". Thus version checks are done
in
         * constant time and are treated like any other decryption error.
         */
        version_good =
            constant_time_eq_8(p[0], (unsigned)(s->client_version >> 8));
        version_good &=
            constant_time_eq_8(p[1], (unsigned)(s->client_version &
0xff));

        /*
         * The premaster secret must contain the same version number as
the
         * ClientHello to detect version rollback attacks (strangely, the
         * protocol does not offer such protection for DH ciphersuites).
         * However, buggy clients exist that send the negotiated protocol
         * version instead if the server does not support the requested
         * protocol version. If SSL_OP_TLS_ROLLBACK_BUG is set, tolerate
such
         * clients.
         */
        if (s->options & SSL_OP_TLS_ROLLBACK_BUG) {
            unsigned char workaround_good;
            workaround_good =
                constant_time_eq_8(p[0], (unsigned)(s->version >> 8));
            workaround_good &=
                constant_time_eq_8(p[1], (unsigned)(s->version & 0xff));
```

```
            version_good |= workaround_good;
        }

        /*
         * Both decryption and version must be good for decrypt_good to
         * remain non-zero (0xff).
         */
        decrypt_good &= version_good;

        /*
         * Now copy rand_premaster_secret over from p using
         * decrypt_good_mask. If decryption failed, then p does not
         * contain valid plaintext, however, a check above guarantees
         * it is still sufficiently large to read from.
         */
        for (j = 0; j < sizeof(rand_premaster_secret); j++) {
            p[j] = constant_time_select_8(decrypt_good, p[j],
                                          rand_premaster_secret[j]);
        }

        s->session->master_key_length =
            s->method->ssl3_enc->generate_master_secret(s,
                                                        s->
                                                        session-
>master_key,
                                                        p,
                                                        sizeof

(rand_premaster_secret));
        OPENSSL_cleanse(p, sizeof(rand_premaster_secret));
    } else
#endif
#ifndef OPENSSL_NO_DH
    if (alg_k & (SSL_kEDH | SSL_kDHr | SSL_kDHd)) {
        int idx = -1;
        EVP_PKEY *skey = NULL;
        if (n)
            n2s(p, i);
        else
            i = 0;
        if (n && n != i + 2) {
            if (!(s->options & SSL_OP_SSLEAY_080_CLIENT_DH_BUG)) {
                SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                       SSL_R_DH_PUBLIC_VALUE_LENGTH_IS_WRONG);
                goto err;
            } else {
                p -= 2;
                i = (int)n;
            }
        }
        if (alg_k & SSL_kDHr)
            idx = SSL_PKEY_DH_RSA;
        else if (alg_k & SSL_kDHd)
            idx = SSL_PKEY_DH_DSA;
```

```
if (idx >= 0) {
    skey = s->cert->pkeys[idx].privatekey;
    if ((skey == NULL) ||
        (skey->type != EVP_PKEY_DH) || (skey->pkey.dh == NULL)) {
        al = SSL_AD_HANDSHAKE_FAILURE;
        SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
               SSL_R_MISSING_RSA_CERTIFICATE);
        goto f_err;
    }
    dh_srvr = skey->pkey.dh;
} else if (s->s3->tmp.dh == NULL) {
    al = SSL_AD_HANDSHAKE_FAILURE;
    SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
           SSL_R_MISSING_TMP_DH_KEY);
    goto f_err;
} else
    dh_srvr = s->s3->tmp.dh;

if (n == 0L) {
    /* Get pubkey from cert */
    EVP_PKEY *clkey = X509_get_pubkey(s->session->peer);
    if (clkey) {
        if (EVP_PKEY_cmp_parameters(clkey, skey) == 1)
            dh_clnt = EVP_PKEY_get1_DH(clkey);
    }
    if (dh_clnt == NULL) {
        al = SSL_AD_HANDSHAKE_FAILURE;
        SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
               SSL_R_MISSING_TMP_DH_KEY);
        goto f_err;
    }
    EVP_PKEY_free(clkey);
    pub = dh_clnt->pub_key;
} else
    pub = BN_bin2bn(p, i, NULL);
if (pub == NULL) {
    SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE, SSL_R_BN_LIB);
    goto err;
}

i = DH_compute_key(p, pub, dh_srvr);

if (i <= 0) {
    SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE, ERR_R_DH_LIB);
    BN_clear_free(pub);
    goto err;
}

DH_free(s->s3->tmp.dh);
s->s3->tmp.dh = NULL;
if (dh_clnt)
    DH_free(dh_clnt);
else
    BN_clear_free(pub);
```

```
        pub = NULL;
        s->session->master_key_length =
            s->method->ssl3_enc->generate_master_secret(s,
                                                        s->
                                                        session-
>master_key,
                                                        p, i);
        OPENSSL_cleanse(p, i);
        if (dh_clnt)
            return 2;
    } else
#endif
#ifndef OPENSSL_NO_KRB5
    if (alg_k & SSL_kKRB5) {
        krb5_error_code krb5rc;
        krb5_data enc_ticket;
        krb5_data authenticator;
        krb5_data enc_pms;
        KSSL_CTX *kssl_ctx = s->kssl_ctx;
        EVP_CIPHER_CTX ciph_ctx;
        const EVP_CIPHER *enc = NULL;
        unsigned char iv[EVP_MAX_IV_LENGTH];
        unsigned char pms[SSL_MAX_MASTER_KEY_LENGTH +
EVP_MAX_BLOCK_LENGTH];
        int padl, outl;
        krb5_timestamp authtime = 0;
        krb5_ticket_times ttimes;

        EVP_CIPHER_CTX_init(&ciph_ctx);

        if (!kssl_ctx)
            kssl_ctx = kssl_ctx_new();

        n2s(p, i);
        enc_ticket.length = i;

        if (n < (long)(enc_ticket.length + 6)) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_DATA_LENGTH_TOO_LONG);
            goto err;
        }

        enc_ticket.data = (char *)p;
        p += enc_ticket.length;

        n2s(p, i);
        authenticator.length = i;

        if (n < (long)(enc_ticket.length + authenticator.length + 6)) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_DATA_LENGTH_TOO_LONG);
            goto err;
        }
```

```
        authenticator.data = (char *)p;
        p += authenticator.length;

        n2s(p, i);
        enc_pms.length = i;
        enc_pms.data = (char *)p;
        p += enc_pms.length;

        /*
         * Note that the length is checked again below, ** after
decryption
         */
        if (enc_pms.length > sizeof pms) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_DATA_LENGTH_TOO_LONG);
            goto err;
        }

        if (n != (long)(enc_ticket.length + authenticator.length +
                        enc_pms.length + 6)) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_DATA_LENGTH_TOO_LONG);
            goto err;
        }

        if ((krb5rc = kssl_sget_tkt(kssl_ctx, &enc_ticket, &ttimes,
                                    &kssl_err)) != 0) {
# ifdef KSSL_DEBUG
            fprintf(stderr, "kssl_sget_tkt rtn %d [%d]\n",
                    krb5rc, kssl_err.reason);
            if (kssl_err.text)
                fprintf(stderr, "kssl_err text= %s\n", kssl_err.text);
# endif                         /* KSSL_DEBUG */
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE, kssl_err.reason);
            goto err;
        }

        /*
         * Note: no authenticator is not considered an error, ** but will
         * return authtime == 0.
         */
        if ((krb5rc = kssl_check_authent(kssl_ctx, &authenticator,
                                         &authtime, &kssl_err)) != 0) {
# ifdef KSSL_DEBUG
            fprintf(stderr, "kssl_check_authent rtn %d [%d]\n",
                    krb5rc, kssl_err.reason);
            if (kssl_err.text)
                fprintf(stderr, "kssl_err text= %s\n", kssl_err.text);
# endif                         /* KSSL_DEBUG */
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE, kssl_err.reason);
            goto err;
        }

        if ((krb5rc = kssl_validate_times(authtime, &ttimes)) != 0) {
```

```
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE, krb5rc);
            goto err;
        }
# ifdef KSSL_DEBUG
        kssl_ctx_show(kssl_ctx);
# endif                         /* KSSL_DEBUG */

        enc = kssl_map_enc(kssl_ctx->enctype);
        if (enc == NULL)
            goto err;

        memset(iv, 0, sizeof iv); /* per RFC 1510 */

        if (!EVP_DecryptInit_ex(&ciph_ctx, enc, NULL, kssl_ctx->key, iv))
{
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_DECRYPTION_FAILED);
            goto err;
        }
        if (!EVP_DecryptUpdate(&ciph_ctx, pms, &outl,
                               (unsigned char *)enc_pms.data,
enc_pms.length))
        {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_DECRYPTION_FAILED);
            goto err;
        }
        if (outl > SSL_MAX_MASTER_KEY_LENGTH) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_DATA_LENGTH_TOO_LONG);
            goto err;
        }
        if (!EVP_DecryptFinal_ex(&ciph_ctx, &(pms[outl]), &padl)) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_DECRYPTION_FAILED);
            goto err;
        }
        outl += padl;
        if (outl > SSL_MAX_MASTER_KEY_LENGTH) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_DATA_LENGTH_TOO_LONG);
            goto err;
        }
        if (!((pms[0] == (s->client_version >> 8))
              && (pms[1] == (s->client_version & 0xff)))) {
            /*
             * The premaster secret must contain the same version number
as
             * the ClientHello to detect version rollback attacks
(strangely,
             * the protocol does not offer such protection for DH
             * ciphersuites). However, buggy clients exist that send
random
             * bytes instead of the protocol version. If
```

```
             * SSL_OP_TLS_ROLLBACK_BUG is set, tolerate such clients.
             * (Perhaps we should have a separate BUG value for the
Kerberos
             * cipher)
             */
            if (!(s->options & SSL_OP_TLS_ROLLBACK_BUG)) {
                SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                       SSL_AD_DECODE_ERROR);
                goto err;
            }
        }

        EVP_CIPHER_CTX_cleanup(&ciph_ctx);

        s->session->master_key_length =
            s->method->ssl3_enc->generate_master_secret(s,
                                                        s->
                                                        session-
>master_key,
                                                        pms, outl);

        if (kssl_ctx->client_princ) {
            size_t len = strlen(kssl_ctx->client_princ);
            if (len < SSL_MAX_KRB5_PRINCIPAL_LENGTH) {
                s->session->krb5_client_princ_len = len;
                memcpy(s->session->krb5_client_princ, kssl_ctx-
>client_princ,
                       len);
            }
        }

        /*- Was doing kssl_ctx_free() here,
         *  but it caused problems for apache.
         *  kssl_ctx = kssl_ctx_free(kssl_ctx);
         *  if (s->kssl_ctx)  s->kssl_ctx = NULL;
         */
    } else
#endif                          /* OPENSSL_NO_KRB5 */

#ifndef OPENSSL_NO_ECDH
    if (alg_k & (SSL_kEECDH | SSL_kECDHr | SSL_kECDHe)) {
        int ret = 1;
        int field_size = 0;
        const EC_KEY *tkey;
        const EC_GROUP *group;
        const BIGNUM *priv_key;

        /* initialize structures for server's ECDH key pair */
        if ((srvr_ecdh = EC_KEY_new()) == NULL) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
ERR_R_MALLOC_FAILURE);
            goto err;
        }
```

```
        /* Let's get server private key and group information */
        if (alg_k & (SSL_kECDHr | SSL_kECDHe)) {
            /* use the certificate */
            tkey = s->cert->pkeys[SSL_PKEY_ECC].privatekey->pkey.ec;
        } else {
            /*
             * use the ephermeral values we saved when generating the
             * ServerKeyExchange msg.
             */
            tkey = s->s3->tmp.ecdh;
        }

        group = EC_KEY_get0_group(tkey);
        priv_key = EC_KEY_get0_private_key(tkey);

        if (!EC_KEY_set_group(srvr_ecdh, group) ||
            !EC_KEY_set_private_key(srvr_ecdh, priv_key)) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE, ERR_R_EC_LIB);
            goto err;
        }

        /* Let's get client's public key */
        if ((clnt_ecpoint = EC_POINT_new(group)) == NULL) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
ERR_R_MALLOC_FAILURE);
            goto err;
        }

        if (n == 0L) {
            /* Client Publickey was in Client Certificate */

            if (alg_k & SSL_kEECDH) {
                al = SSL_AD_HANDSHAKE_FAILURE;
                SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                       SSL_R_MISSING_TMP_ECDH_KEY);
                goto f_err;
            }
            if (((clnt_pub_pkey = X509_get_pubkey(s->session->peer))
                 == NULL) || (clnt_pub_pkey->type != EVP_PKEY_EC)) {
                /*
                 * XXX: For now, we do not support client authentication
                 * using ECDH certificates so this branch (n == 0L) of
the
                 * code is never executed. When that support is added, we
                 * ought to ensure the key received in the certificate is
                 * authorized for key agreement. ECDH_compute_key
implicitly
                 * checks that the two ECDH shares are for the same
group.
                 */
                al = SSL_AD_HANDSHAKE_FAILURE;
                SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                       SSL_R_UNABLE_TO_DECODE_ECDH_CERTS);
                goto f_err;
```

```
            }

            if (EC_POINT_copy(clnt_ecpoint,
                              EC_KEY_get0_public_key(clnt_pub_pkey->
                                                     pkey.ec)) == 0) {
                SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE, ERR_R_EC_LIB);
                goto err;
            }
            ret = 2;            /* Skip certificate verify processing */
        } else {
            /*
             * Get client's public key from encoded point in the
             * ClientKeyExchange message.
             */
            if ((bn_ctx = BN_CTX_new()) == NULL) {
                SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                       ERR_R_MALLOC_FAILURE);
                goto err;
            }

            /* Get encoded point length */
            i = *p;
            p += 1;
            if (n != 1 + i) {
                SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE, ERR_R_EC_LIB);
                goto err;
            }
            if (EC_POINT_oct2point(group, clnt_ecpoint, p, i, bn_ctx) ==
0) {
                SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE, ERR_R_EC_LIB);
                goto err;
            }
            /*
             * p is pointing to somewhere in the buffer currently, so set
it
             * to the start
             */
            p = (unsigned char *)s->init_buf->data;
        }

        /* Compute the shared pre-master secret */
        field_size = EC_GROUP_get_degree(group);
        if (field_size <= 0) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE, ERR_R_ECDH_LIB);
            goto err;
        }
        i = ECDH_compute_key(p, (field_size + 7) / 8, clnt_ecpoint,
srvr_ecdh,
                             NULL);
        if (i <= 0) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE, ERR_R_ECDH_LIB);
            goto err;
        }
```

```
        EVP_PKEY_free(clnt_pub_pkey);
        EC_POINT_free(clnt_ecpoint);
        EC_KEY_free(srvr_ecdh);
        BN_CTX_free(bn_ctx);
        EC_KEY_free(s->s3->tmp.ecdh);
        s->s3->tmp.ecdh = NULL;

        /* Compute the master secret */
        s->session->master_key_length =
            s->method->ssl3_enc->generate_master_secret(s,
                                                        s->
                                                        session-
>master_key,
                                                        p, i);

        OPENSSL_cleanse(p, i);
        return (ret);
    } else
#endif
#ifndef OPENSSL_NO_PSK
    if (alg_k & SSL_kPSK) {
        unsigned char *t = NULL;
        unsigned char psk_or_pre_ms[PSK_MAX_PSK_LEN * 2 + 4];
        unsigned int pre_ms_len = 0, psk_len = 0;
        int psk_err = 1;
        char tmp_id[PSK_MAX_IDENTITY_LEN + 1];

        al = SSL_AD_HANDSHAKE_FAILURE;

        n2s(p, i);
        if (n != i + 2) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
SSL_R_LENGTH_MISMATCH);
            goto psk_err;
        }
        if (i > PSK_MAX_IDENTITY_LEN) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_DATA_LENGTH_TOO_LONG);
            goto psk_err;
        }
        if (s->psk_server_callback == NULL) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_PSK_NO_SERVER_CB);
            goto psk_err;
        }

        /*
         * Create guaranteed NULL-terminated identity string for the
callback
         */
        memcpy(tmp_id, p, i);
        memset(tmp_id + i, 0, PSK_MAX_IDENTITY_LEN + 1 - i);
        psk_len = s->psk_server_callback(s, tmp_id,
                                         psk_or_pre_ms,
```

```
                                        sizeof(psk_or_pre_ms));
        OPENSSL_cleanse(tmp_id, PSK_MAX_IDENTITY_LEN + 1);

        if (psk_len > PSK_MAX_PSK_LEN) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
ERR_R_INTERNAL_ERROR);
            goto psk_err;
        } else if (psk_len == 0) {
            /*
             * PSK related to the given identity not found
             */
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_PSK_IDENTITY_NOT_FOUND);
            al = SSL_AD_UNKNOWN_PSK_IDENTITY;
            goto psk_err;
        }

        /* create PSK pre_master_secret */
        pre_ms_len = 2 + psk_len + 2 + psk_len;
        t = psk_or_pre_ms;
        memmove(psk_or_pre_ms + psk_len + 4, psk_or_pre_ms, psk_len);
        s2n(psk_len, t);
        memset(t, 0, psk_len);
        t += psk_len;
        s2n(psk_len, t);

        if (s->session->psk_identity != NULL)
            OPENSSL_free(s->session->psk_identity);
        s->session->psk_identity = BUF_strdup((char *)p);
        if (s->session->psk_identity == NULL) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
ERR_R_MALLOC_FAILURE);
            goto psk_err;
        }

        if (s->session->psk_identity_hint != NULL)
            OPENSSL_free(s->session->psk_identity_hint);
        s->session->psk_identity_hint = BUF_strdup(s->ctx-
>psk_identity_hint);
        if (s->ctx->psk_identity_hint != NULL &&
            s->session->psk_identity_hint == NULL) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
ERR_R_MALLOC_FAILURE);
            goto psk_err;
        }

        s->session->master_key_length =
            s->method->ssl3_enc->generate_master_secret(s,
                                                        s->
                                                        session-
>master_key,
                                                        psk_or_pre_ms,
                                                        pre_ms_len);
        psk_err = 0;
```

```
 psk_err:
        OPENSSL_cleanse(psk_or_pre_ms, sizeof(psk_or_pre_ms));
        if (psk_err != 0)
            goto f_err;
    } else
#endif
#ifndef OPENSSL_NO_SRP
    if (alg_k & SSL_kSRP) {
        int param_len;

        n2s(p, i);
        param_len = i + 2;
        if (param_len > n) {
            al = SSL_AD_DECODE_ERROR;
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_BAD_SRP_A_LENGTH);
            goto f_err;
        }
        if (!(s->srp_ctx.A = BN_bin2bn(p, i, NULL))) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE, ERR_R_BN_LIB);
            goto err;
        }
        if (BN_ucmp(s->srp_ctx.A, s->srp_ctx.N) >= 0
            || BN_is_zero(s->srp_ctx.A)) {
            al = SSL_AD_ILLEGAL_PARAMETER;
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_BAD_SRP_PARAMETERS);
            goto f_err;
        }
        if (s->session->srp_username != NULL)
            OPENSSL_free(s->session->srp_username);
        s->session->srp_username = BUF_strdup(s->srp_ctx.login);
        if (s->session->srp_username == NULL) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
ERR_R_MALLOC_FAILURE);
            goto err;
        }

        if ((s->session->master_key_length =
             SRP_generate_server_master_secret(s,
                                               s->session->master_key)) <
0) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
ERR_R_INTERNAL_ERROR);
            goto err;
        }

        p += i;
    } else
#endif                          /* OPENSSL_NO_SRP */
    if (alg_k & SSL_kGOST) {
        int ret = 0;
        EVP_PKEY_CTX *pkey_ctx;
        EVP_PKEY *client_pub_pkey = NULL, *pk = NULL;
```

```
        unsigned char premaster_secret[32], *start;
        size_t outlen = 32, inlen;
        unsigned long alg_a;
        int Ttag, Tclass;
        long Tlen;

        /* Get our certificate private key */
        alg_a = s->s3->tmp.new_cipher->algorithm_auth;
        if (alg_a & SSL_aGOST94)
            pk = s->cert->pkeys[SSL_PKEY_GOST94].privatekey;
        else if (alg_a & SSL_aGOST01)
            pk = s->cert->pkeys[SSL_PKEY_GOST01].privatekey;

        pkey_ctx = EVP_PKEY_CTX_new(pk, NULL);
        EVP_PKEY_decrypt_init(pkey_ctx);
        /*
         * If client certificate is present and is of the same type,
maybe
         * use it for key exchange.  Don't mind errors from
         * EVP_PKEY_derive_set_peer, because it is completely valid to
use a
         * client certificate for authorization only.
         */
        client_pub_pkey = X509_get_pubkey(s->session->peer);
        if (client_pub_pkey) {
            if (EVP_PKEY_derive_set_peer(pkey_ctx, client_pub_pkey) <= 0)
                ERR_clear_error();
        }
        /* Decrypt session key */
        if (ASN1_get_object
            ((const unsigned char **)&p, &Tlen, &Ttag, &Tclass,
             n) != V_ASN1_CONSTRUCTED || Ttag != V_ASN1_SEQUENCE
            || Tclass != V_ASN1_UNIVERSAL) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_DECRYPTION_FAILED);
            goto gerr;
        }
        start = p;
        inlen = Tlen;
        if (EVP_PKEY_decrypt
            (pkey_ctx, premaster_secret, &outlen, start, inlen) <= 0) {
            SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
                   SSL_R_DECRYPTION_FAILED);
            goto gerr;
        }
        /* Generate master secret */
        s->session->master_key_length =
            s->method->ssl3_enc->generate_master_secret(s,
                                                        s->
                                                        session-
>master_key,
                                                        premaster_secret,
32);
        /* Check if pubkey from client certificate was used */
```

```
        if (EVP_PKEY_CTX_ctrl
            (pkey_ctx, -1, -1, EVP_PKEY_CTRL_PEER_KEY, 2, NULL) > 0)
            ret = 2;
        else
            ret = 1;
 gerr:
        EVP_PKEY_free(client_pub_pkey);
        EVP_PKEY_CTX_free(pkey_ctx);
        if (ret)
            return ret;
        else
            goto err;
    } else {
        al = SSL_AD_HANDSHAKE_FAILURE;
        SSLerr(SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE,
SSL_R_UNKNOWN_CIPHER_TYPE);
        goto f_err;
    }

    return (1);
 f_err:
    ssl3_send_alert(s, SSL3_AL_FATAL, al);
#if !defined(OPENSSL_NO_DH) || !defined(OPENSSL_NO_RSA) ||
!defined(OPENSSL_NO_ECDH) || defined(OPENSSL_NO_SRP)
 err:
#endif
#ifndef OPENSSL_NO_ECDH
    EVP_PKEY_free(clnt_pub_pkey);
    EC_POINT_free(clnt_ecpoint);
    if (srvr_ecdh != NULL)
        EC_KEY_free(srvr_ecdh);
    BN_CTX_free(bn_ctx);
#endif
    return (-1);
}
<sep>
static void print_qualifiers(BIO *out, STACK_OF(POLICYQUALINFO) *quals,
                             int indent)
{
    POLICYQUALINFO *qualinfo;
    int i;
    for (i = 0; i < sk_POLICYQUALINFO_num(quals); i++) {
        qualinfo = sk_POLICYQUALINFO_value(quals, i);
        switch (OBJ_obj2nid(qualinfo->pqualid)) {
        case NID_id_qt_cps:
            BIO_printf(out, "%*sCPS: %s\n", indent, "",
                       qualinfo->d.cpsuri->data);
            break;

        case NID_id_qt_unotice:
            BIO_printf(out, "%*sUser Notice:\n", indent, "");
            print_notice(out, qualinfo->d.usernotice, indent + 2);
            break;
```

```
        default:
            BIO_printf(out, "%*sUnknown Qualifier: ", indent + 2, "");

            i2a_ASN1_OBJECT(out, qualinfo->pqualid);
            BIO_puts(out, "\n");
            break;
        }
    }
}
<sep>
static void handle_rx(struct vhost_net *net)
{
      struct vhost_net_virtqueue *nvq = &net->vqs[VHOST_NET_VQ_RX];
      struct vhost_virtqueue *vq = &nvq->vq;
      unsigned uninitialized_var(in), log;
      struct vhost_log *vq_log;
      struct msghdr msg = {
            .msg_name = NULL,
            .msg_namelen = 0,
            .msg_control = NULL, /* FIXME: get and handle RX aux data. */
            .msg_controllen = 0,
            .msg_iov = vq->iov,
            .msg_flags = MSG_DONTWAIT,
      };
      struct virtio_net_hdr_mrg_rxbuf hdr = {
            .hdr.flags = 0,
            .hdr.gso_type = VIRTIO_NET_HDR_GSO_NONE
      };
      size_t total_len = 0;
      int err, mergeable;
      s16 headcount;
      size_t vhost_hlen, sock_hlen;
      size_t vhost_len, sock_len;
      struct socket *sock;

      mutex_lock(&vq->mutex);
      sock = vq->private_data;
      if (!sock)
            goto out;
      vhost_disable_notify(&net->dev, vq);

      vhost_hlen = nvq->vhost_hlen;
      sock_hlen = nvq->sock_hlen;

      vq_log = unlikely(vhost_has_feature(&net->dev, VHOST_F_LOG_ALL)) ?
            vq->log : NULL;
      mergeable = vhost_has_feature(&net->dev, VIRTIO_NET_F_MRG_RXBUF);

      while ((sock_len = peek_head_len(sock->sk))) {
            sock_len += sock_hlen;
            vhost_len = sock_len + vhost_hlen;
            headcount = get_rx_bufs(vq, vq->heads, vhost_len,
                              &in, vq_log, &log,
                              likely(mergeable) ? UIO_MAXIOV : 1);
```

```
                /* On error, stop handling until the next kick. */
                if (unlikely(headcount < 0))
                      break;
                /* OK, now we need to know about added descriptors. */
                if (!headcount) {
                      if (unlikely(vhost_enable_notify(&net->dev, vq))) {
                            /* They have slipped one in as we were
                             * doing that: check again. */
                            vhost_disable_notify(&net->dev, vq);
                            continue;
                      }
                      /* Nothing new?  Wait for eventfd to tell us
                       * they refilled. */
                      break;
                }
                /* We don't need to be notified again. */
                if (unlikely((vhost_hlen)))
                      /* Skip header. TODO: support TSO. */
                      move_iovec_hdr(vq->iov, nvq->hdr, vhost_hlen, in);
                else
                      /* Copy the header for use in VIRTIO_NET_F_MRG_RXBUF:
                       * needed because recvmsg can modify msg_iov. */
                      copy_iovec_hdr(vq->iov, nvq->hdr, sock_hlen, in);
               msg.msg_iovlen = in;
               err = sock->ops->recvmsg(NULL, sock, &msg,
                                        sock_len, MSG_DONTWAIT | MSG_TRUNC);
               /* Userspace might have consumed the packet meanwhile:
                * it's not supposed to do this usually, but might be hard
                * to prevent. Discard data we got (if any) and keep going.
*/
               if (unlikely(err != sock_len)) {
                      pr_debug("Discarded rx packet: "
                             " len %d, expected %zd\n", err, sock_len);
                      vhost_discard_vq_desc(vq, headcount);
                      continue;
               }
               if (unlikely(vhost_hlen) &&
                   memcpy_toiovecend(nvq->hdr, (unsigned char *)&hdr, 0,
                                     vhost_hlen)) {
                      vq_err(vq, "Unable to write vnet_hdr at addr %p\n",
                             vq->iov->iov_base);
                      break;
               }
               /* TODO: Should check and handle checksum. */
               if (likely(mergeable) &&
                   memcpy_toiovecend(nvq->hdr, (unsigned char *)&headcount,
                                     offsetof(typeof(hdr), num_buffers),
                                     sizeof hdr.num_buffers)) {
                      vq_err(vq, "Failed num_buffers write");
                      vhost_discard_vq_desc(vq, headcount);
                      break;
               }
               vhost_add_used_and_signal_n(&net->dev, vq, vq->heads,
                                           headcount);
```

```
            if (unlikely(vq_log))
                  vhost_log_write(vq, vq_log, log, vhost_len);
            total_len += vhost_len;
            if (unlikely(total_len >= VHOST_NET_WEIGHT)) {
                  vhost_poll_queue(&vq->poll);
                  break;
            }
      }
out:
      mutex_unlock(&vq->mutex);
}
<sep>
void * pvPortMalloc( size_t xWantedSize )
{
    void * pvReturn = NULL;
    static uint8_t * pucAlignedHeap = NULL;

    /* Ensure that blocks are always aligned to the required number of
bytes. */
    #if ( portBYTE_ALIGNMENT != 1 )
        {
            if( xWantedSize & portBYTE_ALIGNMENT_MASK )
            {
                /* Byte alignment required. */
                xWantedSize += ( portBYTE_ALIGNMENT - ( xWantedSize &
portBYTE_ALIGNMENT_MASK ) );
            }
        }
    #endif

    vTaskSuspendAll();
    {
        if( pucAlignedHeap == NULL )
        {
            /* Ensure the heap starts on a correctly aligned boundary. */
            pucAlignedHeap = ( uint8_t * ) ( ( ( portPOINTER_SIZE_TYPE )
& ucHeap[ portBYTE_ALIGNMENT ] ) & ( ~( ( portPOINTER_SIZE_TYPE )
portBYTE_ALIGNMENT_MASK ) ) );
        }

        /* Check there is enough room left for the allocation. */
        if( ( ( xNextFreeByte + xWantedSize ) < configADJUSTED_HEAP_SIZE
) &&
            ( ( xNextFreeByte + xWantedSize ) > xNextFreeByte ) ) /*
Check for overflow. */
        {
            /* Return the next free byte then increment the index past
this
             * block. */
            pvReturn = pucAlignedHeap + xNextFreeByte;
            xNextFreeByte += xWantedSize;
        }

        traceMALLOC( pvReturn, xWantedSize );
```

```
    }
    ( void ) xTaskResumeAll();

    #if ( configUSE_MALLOC_FAILED_HOOK == 1 )
        {
            if( pvReturn == NULL )
            {
                extern void vApplicationMallocFailedHook( void );
                vApplicationMallocFailedHook();
            }
        }
    #endif

    return pvReturn;
}
<sep>
int mp_unpack_full(lua_State *L, int limit, int offset) {
    size_t len;
    const char *s;
    mp_cur c;
    int cnt; /* Number of objects unpacked */
    int decode_all = (!limit && !offset);

    s = luaL_checklstring(L,1,&len); /* if no match, exits */

    if (offset < 0 || limit < 0) /* requesting negative off or lim is
invalid */
        return luaL_error(L,
            "Invalid request to unpack with offset of %d and limit of
%d.",
            offset, len);
    else if (offset > len)
        return luaL_error(L,
            "Start offset %d greater than input length %d.", offset,
len);

    if (decode_all) limit = INT_MAX;

    mp_cur_init(&c,(const unsigned char *)s+offset,len-offset);

    /* We loop over the decode because this could be a stream
     * of multiple top-level values serialized together */
    for(cnt = 0; c.left > 0 && cnt < limit; cnt++) {
        mp_decode_to_lua_type(L,&c);

        if (c.err == MP_CUR_ERROR_EOF) {
            return luaL_error(L,"Missing bytes in input.");
        } else if (c.err == MP_CUR_ERROR_BADFMT) {
            return luaL_error(L,"Bad data format in input.");
        }
    }

    if (!decode_all) {
        /* c->left is the remaining size of the input buffer.
```

```
         * subtract the entire buffer size from the unprocessed size
         * to get our next start offset */
        int offset = len - c.left;
        /* Return offset -1 when we have have processed the entire
buffer. */
        lua_pushinteger(L, c.left == 0 ? -1 : offset);
        /* Results are returned with the arg elements still
         * in place. Lua takes care of only returning
         * elements above the args for us.
         * In this case, we have one arg on the stack
         * for this function, so we insert our first return
         * value at position 2. */
        lua_insert(L, 2);
        cnt += 1; /* increase return count by one to make room for offset
*/
    }

    return cnt;
}
<sep>
static int nfs_init_server(struct nfs_server *server, const struct
nfs_mount_data *data)
{
      struct nfs_client *clp;
      int error, nfsvers = 2;

      dprintk("--> nfs_init_server()\n");

#ifdef CONFIG_NFS_V3
      if (data->flags & NFS_MOUNT_VER3)
            nfsvers = 3;
#endif

      /* Allocate or find a client reference we can use */
      clp = nfs_get_client(data->hostname, &data->addr, nfsvers);
      if (IS_ERR(clp)) {
            dprintk("<-- nfs_init_server() = error %ld\n", PTR_ERR(clp));
            return PTR_ERR(clp);
      }

      error = nfs_init_client(clp, data);
      if (error < 0)
            goto error;

      server->nfs_client = clp;

      /* Initialise the client representation from the mount data */
      server->flags = data->flags & NFS_MOUNT_FLAGMASK;

      if (data->rsize)
            server->rsize = nfs_block_size(data->rsize, NULL);
      if (data->wsize)
            server->wsize = nfs_block_size(data->wsize, NULL);
```

```
      server->acregmin = data->acregmin * HZ;
      server->acregmax = data->acregmax * HZ;
      server->acdirmin = data->acdirmin * HZ;
      server->acdirmax = data->acdirmax * HZ;

      /* Start lockd here, before we might error out */
      error = nfs_start_lockd(server);
      if (error < 0)
            goto error;

      error = nfs_init_server_rpcclient(server, data->pseudoflavor);
      if (error < 0)
            goto error;

      server->namelen  = data->namlen;
      /* Create a client RPC handle for the NFSv3 ACL management
interface */
      nfs_init_server_aclclient(server);
      if (clp->cl_nfsversion == 3) {
            if (server->namelen == 0 || server->namelen > NFS3_MAXNAMLEN)
                  server->namelen = NFS3_MAXNAMLEN;
            if (!(data->flags & NFS_MOUNT_NORDIRPLUS))
                  server->caps |= NFS_CAP_READDIRPLUS;
      } else {
            if (server->namelen == 0 || server->namelen > NFS2_MAXNAMLEN)
                  server->namelen = NFS2_MAXNAMLEN;
      }

      dprintk("<-- nfs_init_server() = 0 [new %p]\n", clp);
      return 0;

error:
      server->nfs_client = NULL;
      nfs_put_client(clp);
      dprintk("<-- nfs_init_server() = xerror %d\n", error);
      return error;
}
<sep>
PGTYPEStimestamp_from_asc(char *str, char **endptr)
{
      timestamp   result;

#ifdef HAVE_INT64_TIMESTAMP
      int64       noresult = 0;
#else
      double            noresult = 0.0;
#endif
      fsec_t            fsec;
      struct tm   tt,
                     *tm = &tt;
      int               dtype;
      int               nf;
      char     *field[MAXDATEFIELDS];
      int               ftype[MAXDATEFIELDS];
```

```
	char		lowstr[MAXDATELEN + MAXDATEFIELDS];
	char	 *realptr;
	char	**ptr = (endptr != NULL) ? endptr : &realptr;

	if (strlen(str) >= sizeof(lowstr))
	{
		errno = PGTYPES_TS_BAD_TIMESTAMP;
		return (noresult);
	}

	if (ParseDateTime(str, lowstr, field, ftype, &nf, ptr) != 0 ||
		DecodeDateTime(field, ftype, nf, &dtype, tm, &fsec, 0) != 0)
	{
		errno = PGTYPES_TS_BAD_TIMESTAMP;
		return (noresult);
	}

	switch (dtype)
	{
		case DTK_DATE:
			if (tm2timestamp(tm, fsec, NULL, &result) != 0)
			{
				errno = PGTYPES_TS_BAD_TIMESTAMP;
				return (noresult);
			}
			break;

		case DTK_EPOCH:
			result = SetEpochTimestamp();
			break;

		case DTK_LATE:
			TIMESTAMP_NOEND(result);
			break;

		case DTK_EARLY:
			TIMESTAMP_NOBEGIN(result);
			break;

		case DTK_INVALID:
			errno = PGTYPES_TS_BAD_TIMESTAMP;
			return (noresult);

		default:
			errno = PGTYPES_TS_BAD_TIMESTAMP;
			return (noresult);
	}

	/* AdjustTimestampForTypmod(&result, typmod); */

	/*
	 * Since it's difficult to test for noresult, make sure errno is 0
if no
	 * error occurred.
```

```
       */
      errno = 0;
      return result;
}
<sep>
static int build_audio_procunit(struct mixer_build *state, int unitid,
                        void *raw_desc, struct procunit_info *list,
                        char *name)
{
      struct uac_processing_unit_descriptor *desc = raw_desc;
      int num_ins = desc->bNrInPins;
      struct usb_mixer_elem_info *cval;
      struct snd_kcontrol *kctl;
      int i, err, nameid, type, len;
      struct procunit_info *info;
      struct procunit_value_info *valinfo;
      const struct usbmix_name_map *map;
      static struct procunit_value_info default_value_info[] = {
            { 0x01, "Switch", USB_MIXER_BOOLEAN },
            { 0 }
      };
      static struct procunit_info default_info = {
            0, NULL, default_value_info
      };

      if (desc->bLength < 13 || desc->bLength < 13 + num_ins ||
          desc->bLength < num_ins +
uac_processing_unit_bControlSize(desc, state->mixer->protocol)) {
            usb_audio_err(state->chip, "invalid %s descriptor (id %d)\n",
name, unitid);
            return -EINVAL;
      }

      for (i = 0; i < num_ins; i++) {
            err = parse_audio_unit(state, desc->baSourceID[i]);
            if (err < 0)
                  return err;
      }

      type = le16_to_cpu(desc->wProcessType);
      for (info = list; info && info->type; info++)
            if (info->type == type)
                  break;
      if (!info || !info->type)
            info = &default_info;

      for (valinfo = info->values; valinfo->control; valinfo++) {
            __u8 *controls = uac_processing_unit_bmControls(desc, state-
>mixer->protocol);

            if (state->mixer->protocol == UAC_VERSION_1) {
                  if (!(controls[valinfo->control / 8] &
                              (1 << ((valinfo->control % 8) - 1))))
                        continue;
```

```
            } else { /* UAC_VERSION_2/3 */
                  if (!uac_v2v3_control_is_readable(controls[valinfo-
>control / 8],
                                                    valinfo->control))
                        continue;
            }

            map = find_map(state->map, unitid, valinfo->control);
            if (check_ignored_ctl(map))
                  continue;
            cval = kzalloc(sizeof(*cval), GFP_KERNEL);
            if (!cval)
                  return -ENOMEM;
            snd_usb_mixer_elem_init_std(&cval->head, state->mixer,
unitid);
            cval->control = valinfo->control;
            cval->val_type = valinfo->val_type;
            cval->channels = 1;

            if (state->mixer->protocol > UAC_VERSION_1 &&
                !uac_v2v3_control_is_writeable(controls[valinfo->control
/ 8],
                                               valinfo->control))
                  cval->master_readonly = 1;

            /* get min/max values */
            switch (type) {
            case UAC_PROCESS_UP_DOWNMIX: {
                  bool mode_sel = false;

                  switch (state->mixer->protocol) {
                  case UAC_VERSION_1:
                  case UAC_VERSION_2:
                  default:
                        if (cval->control == UAC_UD_MODE_SELECT)
                              mode_sel = true;
                        break;
                  case UAC_VERSION_3:
                        if (cval->control == UAC3_UD_MODE_SELECT)
                              mode_sel = true;
                        break;
                  }

                  if (mode_sel) {
                        __u8 *control_spec =
uac_processing_unit_specific(desc,
                                                     state->mixer->protocol);
                        cval->min = 1;
                        cval->max = control_spec[0];
                        cval->res = 1;
                        cval->initialized = 1;
                        break;
                  }
```

```
                get_min_max(cval, valinfo->min_value);
                break;
        }
        case USB_XU_CLOCK_RATE:
                /*
                 * E-Mu USB 0404/0202/TrackerPre/0204
                 * samplerate control quirk
                 */
                cval->min = 0;
                cval->max = 5;
                cval->res = 1;
                cval->initialized = 1;
                break;
        default:
                get_min_max(cval, valinfo->min_value);
                break;
        }

        kctl = snd_ctl_new1(&mixer_procunit_ctl, cval);
        if (!kctl) {
                kfree(cval);
                return -ENOMEM;
        }
        kctl->private_free = snd_usb_mixer_elem_free;

        if (check_mapped_name(map, kctl->id.name, sizeof(kctl-
>id.name))) {
                /* nothing */ ;
        } else if (info->name) {
                strlcpy(kctl->id.name, info->name, sizeof(kctl-
>id.name));
        } else {
                nameid = uac_processing_unit_iProcessing(desc, state-
>mixer->protocol);
                len = 0;
                if (nameid)
                        len = snd_usb_copy_string_desc(state->chip,
                                                       nameid,
                                                       kctl->id.name,
                                                       sizeof(kctl->id.name));
                if (!len)
                        strlcpy(kctl->id.name, name, sizeof(kctl-
>id.name));
        }
        append_ctl_name(kctl, " ");
        append_ctl_name(kctl, valinfo->suffix);

        usb_audio_dbg(state->chip,
                      "[%d] PU [%s] ch = %d, val = %d/%d\n",
                      cval->head.id, kctl->id.name, cval->channels,
                      cval->min, cval->max);

        err = snd_usb_mixer_add_control(&cval->head, kctl);
        if (err < 0)
```

```
                return err;
      }
      return 0;
}
<sep>
static RzList *entries(RzBinFile *bf) {
      if (!bf) {
            return NULL;
      }
      LuacBinInfo *bin_info_obj = GET_INTERNAL_BIN_INFO_OBJ(bf);
      if (!bin_info_obj) {
            return NULL;
      }

      return bin_info_obj->entry_list;
}
<sep>
static inline void eventpoll_init_file(struct file *file)
{
      INIT_LIST_HEAD(&file->f_ep_links);
}
<sep>
lexer_parse_string (parser_context_t *context_p, /**< context */
                    lexer_string_options_t opts) /**< options */
{
#if JERRY_ESNEXT
  int32_t raw_length_adjust = 0;
#else /* JERRY_ESNEXT */
  JERRY_UNUSED (opts);
#endif /* JERRY_ESNEXT */

  uint8_t str_end_character = context_p->source_p[0];
  const uint8_t *source_p = context_p->source_p + 1;
  const uint8_t *string_start_p = source_p;
  const uint8_t *source_end_p = context_p->source_end_p;
  parser_line_counter_t line = context_p->line;
  parser_line_counter_t column = (parser_line_counter_t) (context_p-
>column + 1);
  parser_line_counter_t original_line = line;
  parser_line_counter_t original_column = column;
  size_t length = 0;
  uint8_t has_escape = false;

#if JERRY_ESNEXT
  if (str_end_character == LIT_CHAR_RIGHT_BRACE)
  {
    str_end_character = LIT_CHAR_GRAVE_ACCENT;
  }
#endif /* JERRY_ESNEXT */

  while (true)
  {
    if (source_p >= source_end_p)
    {
```

```
      context_p->token.line = original_line;
      context_p->token.column = (parser_line_counter_t) (original_column
- 1);
      parser_raise_error (context_p, PARSER_ERR_UNTERMINATED_STRING);
    }

    if (*source_p == str_end_character)
    {
      break;
    }

    if (*source_p == LIT_CHAR_BACKSLASH)
    {
      source_p++;
      column++;
      if (source_p >= source_end_p)
      {
        /* Will throw an unterminated string error. */
        continue;
      }

      has_escape = true;

      /* Newline is ignored. */
      if (*source_p == LIT_CHAR_CR)
      {
        source_p++;
        if (source_p < source_end_p
            && *source_p == LIT_CHAR_LF)
        {
#if JERRY_ESNEXT
          raw_length_adjust--;
#endif /* JERRY_ESNEXT */
          source_p++;
        }

        line++;
        column = 1;
        continue;
      }
      else if (*source_p == LIT_CHAR_LF)
      {
        source_p++;
        line++;
        column = 1;
        continue;
      }
      else if (*source_p == LEXER_NEWLINE_LS_PS_BYTE_1 &&
LEXER_NEWLINE_LS_PS_BYTE_23 (source_p))
      {
        source_p += 3;
        line++;
        column = 1;
        continue;
```

```
      }

#if JERRY_ESNEXT
      if (opts & LEXER_STRING_RAW)
      {
        if ((*source_p == LIT_CHAR_GRAVE_ACCENT) || (*source_p ==
LIT_CHAR_BACKSLASH))
        {
          source_p++;
          column++;
          length++;
        }
        continue;
      }
#endif /* JERRY_ESNEXT */

      if (*source_p == LIT_CHAR_0
          && source_p + 1 < source_end_p
          && (*(source_p + 1) < LIT_CHAR_0 || *(source_p + 1) >
LIT_CHAR_9))
      {
        source_p++;
        column++;
        length++;
        continue;
      }

      /* Except \x, \u, and octal numbers, everything is
       * converted to a character which has the same byte length. */
      if (*source_p >= LIT_CHAR_0 && *source_p <= LIT_CHAR_3)
      {
#if JERRY_ESNEXT
        if (str_end_character == LIT_CHAR_GRAVE_ACCENT)
        {
          parser_raise_error (context_p,
PARSER_ERR_TEMPLATE_STR_OCTAL_ESCAPE);
        }
#endif

        if (context_p->status_flags & PARSER_IS_STRICT)
        {
          parser_raise_error (context_p,
PARSER_ERR_OCTAL_ESCAPE_NOT_ALLOWED);
        }

        source_p++;
        column++;

        if (source_p < source_end_p && *source_p >= LIT_CHAR_0 &&
*source_p <= LIT_CHAR_7)
        {
          source_p++;
          column++;
```

```
          if (source_p < source_end_p && *source_p >= LIT_CHAR_0 &&
*source_p <= LIT_CHAR_7)
          {
            /* Numbers >= 0x200 (0x80) requires
             * two bytes for encoding in UTF-8. */
            if (source_p[-2] >= LIT_CHAR_2)
            {
              length++;
            }

            source_p++;
            column++;
          }
        }

        length++;
        continue;
      }

      if (*source_p >= LIT_CHAR_4 && *source_p <= LIT_CHAR_7)
      {
        if (context_p->status_flags & PARSER_IS_STRICT)
        {
          parser_raise_error (context_p,
PARSER_ERR_OCTAL_ESCAPE_NOT_ALLOWED);
        }

        source_p++;
        column++;

        if (source_p < source_end_p && *source_p >= LIT_CHAR_0 &&
*source_p <= LIT_CHAR_7)
        {
          source_p++;
          column++;
        }

        /* The maximum number is 0x4d so the UTF-8
         * representation is always one byte. */
        length++;
        continue;
      }

      if (*source_p == LIT_CHAR_LOWERCASE_X || *source_p ==
LIT_CHAR_LOWERCASE_U)
      {
        uint32_t escape_length = (*source_p == LIT_CHAR_LOWERCASE_X) ? 3
: 5;
        lit_code_point_t code_point = UINT32_MAX;

#if JERRY_ESNEXT
        if (source_p + 4 <= source_end_p
            && source_p[0] == LIT_CHAR_LOWERCASE_U
            && source_p[1] == LIT_CHAR_LEFT_BRACE)
```

```
        {
          code_point = lexer_hex_in_braces_to_code_point (source_p + 2,
source_end_p, &escape_length);
          escape_length--;
        }
        else
        {
#endif /* JERRY_ESNEXT */
          if (source_p + escape_length <= source_end_p)
          {
            code_point = lexer_hex_to_code_point (source_p + 1,
escape_length - 1);
          }
#if JERRY_ESNEXT
        }
#endif /* JERRY_ESNEXT */

        if (code_point == UINT32_MAX)
        {
          context_p->token.line = line;
          context_p->token.column = (parser_line_counter_t) (column - 1);
          parser_raise_error (context_p,
PARSER_ERR_INVALID_UNICODE_ESCAPE_SEQUENCE);
        }

        length += lit_code_point_get_cesu8_length (code_point);

        source_p += escape_length;
        PARSER_PLUS_EQUAL_LC (column, escape_length);
        continue;
      }
    }
#if JERRY_ESNEXT
    else if (str_end_character == LIT_CHAR_GRAVE_ACCENT &&
             source_p[0] == LIT_CHAR_DOLLAR_SIGN &&
             source_p + 1 < source_end_p &&
             source_p[1] == LIT_CHAR_LEFT_BRACE)
    {
      raw_length_adjust--;
      source_p++;
      break;
    }
#endif /* JERRY_ESNEXT */

    if (*source_p >= LIT_UTF8_4_BYTE_MARKER)
    {
      /* Processing 4 byte unicode sequence (even if it is
       * after a backslash). Always converted to two 3 byte
       * long sequence. */
      length += 2 * 3;
      has_escape = true;
      source_p += 4;
#if JERRY_ESNEXT
      raw_length_adjust += 2;
```

```
#endif /* JERRY_ESNEXT */
      column++;
      continue;
    }
    else if (*source_p == LIT_CHAR_TAB)
    {
      column = align_column_to_tab (column);
      /* Subtract -1 because column is increased below. */
      column--;
    }
#if JERRY_ESNEXT
    else if (*source_p == LEXER_NEWLINE_LS_PS_BYTE_1 &&
LEXER_NEWLINE_LS_PS_BYTE_23 (source_p))
    {
      source_p += 3;
      length += 3;
      line++;
      column = 1;
      continue;
    }
    else if (str_end_character == LIT_CHAR_GRAVE_ACCENT)
    {
      /* Newline (without backslash) is part of the string.
         Note: ECMAScript v6, 11.8.6.1 <CR> or <CR><LF> are both
normalized to <LF> */
      if (*source_p == LIT_CHAR_CR)
      {
        has_escape = true;
        source_p++;
        length++;
        if (source_p < source_end_p
            && *source_p == LIT_CHAR_LF)
        {
          source_p++;
          raw_length_adjust--;
        }
        line++;
        column = 1;
        continue;
      }
      else if (*source_p == LIT_CHAR_LF)
      {
        source_p++;
        length++;
        line++;
        column = 1;
        continue;
      }
    }
#endif /* JERRY_ESNEXT */
    else if (*source_p == LIT_CHAR_CR
#if !JERRY_ESNEXT
             || (*source_p == LEXER_NEWLINE_LS_PS_BYTE_1 &&
LEXER_NEWLINE_LS_PS_BYTE_23 (source_p))
```

```
#endif /* !JERRY_ESNEXT */
             || *source_p == LIT_CHAR_LF)
    {
      context_p->token.line = line;
      context_p->token.column = column;
      parser_raise_error (context_p, PARSER_ERR_NEWLINE_NOT_ALLOWED);
    }

    source_p++;
    column++;
    length++;

    while (source_p < source_end_p
           && IS_UTF8_INTERMEDIATE_OCTET (*source_p))
    {
      source_p++;
      length++;
    }
  }

#if JERRY_ESNEXT
  if (opts & LEXER_STRING_RAW)
  {
    length = (size_t) ((source_p - string_start_p) + raw_length_adjust);
  }
#endif /* JERRY_ESNEXT */

  if (length > PARSER_MAXIMUM_STRING_LENGTH)
  {
    parser_raise_error (context_p, PARSER_ERR_STRING_TOO_LONG);
  }

#if JERRY_ESNEXT
  context_p->token.type = ((str_end_character != LIT_CHAR_GRAVE_ACCENT) ?
LEXER_LITERAL
                                                                        :
LEXER_TEMPLATE_LITERAL);
#else /* !JERRY_ESNEXT */
  context_p->token.type = LEXER_LITERAL;
#endif /* JERRY_ESNEXT */

  /* Fill literal data. */
  context_p->token.lit_location.char_p = string_start_p;
  context_p->token.lit_location.length = (prop_length_t) length;
  context_p->token.lit_location.type = LEXER_STRING_LITERAL;
  context_p->token.lit_location.has_escape = has_escape;

  context_p->source_p = source_p + 1;
  context_p->line = line;
  context_p->column = (parser_line_counter_t) (column + 1);
} /* lexer_parse_string */
<sep>
static int parse_token(char **name, char **value, char **cp)
{
```

```
      char *end;

      if (!name || !value || !cp)
            return -BLKID_ERR_PARAM;

      if (!(*value = strchr(*cp, '=')))
            return 0;

      **value = '\0';
      *name = strip_line(*cp);
      *value = skip_over_blank(*value + 1);

      if (**value == '"') {
            end = strchr(*value + 1, '"');
            if (!end) {
                  DBG(READ, ul_debug("unbalanced quotes at: %s", *value));
                  *cp = *value;
                  return -BLKID_ERR_CACHE;
            }
            (*value)++;
            *end = '\0';
            end++;
      } else {
            end = skip_over_word(*value);
            if (*end) {
                  *end = '\0';
                  end++;
            }
      }
      *cp = end;

      return 1;
}
<sep>
export_desktop_file (const char         *app,
                     const char         *branch,
                     const char         *arch,
                     GKeyFile           *metadata,
                     const char * const *previous_ids,
                     int                 parent_fd,
                     const char         *name,
                     struct stat        *stat_buf,
                     char              **target,
                     GCancellable       *cancellable,
                     GError            **error)
{
  gboolean ret = FALSE;
  glnx_autofd int desktop_fd = -1;
  g_autofree char *tmpfile_name = g_strdup_printf ("export-desktop-
XXXXXX");
  g_autoptr(GOutputStream) out_stream = NULL;
  g_autofree gchar *data = NULL;
  gsize data_len;
  g_autofree gchar *new_data = NULL;
```

```
  gsize new_data_len;
  g_autoptr(GKeyFile) keyfile = NULL;
  g_autofree gchar *old_exec = NULL;
  gint old_argc;
  g_auto(GStrv) old_argv = NULL;
  g_auto(GStrv) groups = NULL;
  GString *new_exec = NULL;
  g_autofree char *escaped_app = maybe_quote (app);
  g_autofree char *escaped_branch = maybe_quote (branch);
  g_autofree char *escaped_arch = maybe_quote (arch);
  int i;

  if (!flatpak_openat_noatime (parent_fd, name, &desktop_fd, cancellable,
error))
    goto out;

  if (!read_fd (desktop_fd, stat_buf, &data, &data_len, error))
    goto out;

  keyfile = g_key_file_new ();
  if (!g_key_file_load_from_data (keyfile, data, data_len,
G_KEY_FILE_KEEP_TRANSLATIONS, error))
    goto out;

  if (g_str_has_suffix (name, ".service"))
    {
      g_autofree gchar *dbus_name = NULL;
      g_autofree gchar *expected_dbus_name = g_strndup (name, strlen
(name) - strlen (".service"));

      dbus_name = g_key_file_get_string (keyfile, "D-BUS Service",
"Name", NULL);

      if (dbus_name == NULL || strcmp (dbus_name, expected_dbus_name) !=
0)
        {
          return flatpak_fail_error (error, FLATPAK_ERROR_EXPORT_FAILED,
                                     _("D-Bus service file '%s' has wrong
name"), name);
        }
    }

  if (g_str_has_suffix (name, ".desktop"))
    {
      gsize length;
      g_auto(GStrv) tags = g_key_file_get_string_list (metadata,
                                                       "Application",
                                                       "tags", &length,
                                                       NULL);

      if (tags != NULL)
        {
          g_key_file_set_string_list (keyfile,
                                      G_KEY_FILE_DESKTOP_GROUP,
```

```
                                        "X-Flatpak-Tags",
                                        (const char * const *) tags,
length);
        }

      /* Add a marker so consumers can easily find out that this launches
a sandbox */
      g_key_file_set_string (keyfile, G_KEY_FILE_DESKTOP_GROUP, "X-
Flatpak", app);

      /* If the app has been renamed, add its old .desktop filename to
       * X-Flatpak-RenamedFrom in the new .desktop file, taking care not
to
       * introduce duplicates.
       */
      if (previous_ids != NULL)
        {
          const char *X_FLATPAK_RENAMED_FROM = "X-Flatpak-RenamedFrom";
          g_auto(GStrv) renamed_from = g_key_file_get_string_list
(keyfile,

G_KEY_FILE_DESKTOP_GROUP,

X_FLATPAK_RENAMED_FROM,
                                                                NULL,
NULL);
          g_autoptr(GPtrArray) merged = g_ptr_array_new_with_free_func
(g_free);
          g_autoptr(GHashTable) seen = g_hash_table_new (g_str_hash,
g_str_equal);
          const char *new_suffix;

          for (i = 0; renamed_from != NULL && renamed_from[i] != NULL;
i++)
            {
              if (!g_hash_table_contains (seen, renamed_from[i]))
                {
                  gchar *copy = g_strdup (renamed_from[i]);
                  g_hash_table_insert (seen, copy, copy);
                  g_ptr_array_add (merged, g_steal_pointer (&copy));
                }
            }

          /* If an app was renamed from com.example.Foo to
net.example.Bar, and
           * the new version exports net.example.Bar-suffix.desktop, we
assume the
           * old version exported com.example.Foo-suffix.desktop.
           *
           * This assertion is true because
           * flatpak_name_matches_one_wildcard_prefix() is called on all
           * exported files before we get here.
           */
          g_assert (g_str_has_prefix (name, app));
```

```
          /* ".desktop" for the "main" desktop file; something like
           * "-suffix.desktop" for extra ones.
           */
          new_suffix = name + strlen (app);

          for (i = 0; previous_ids[i] != NULL; i++)
            {
              g_autofree gchar *previous_desktop = g_strconcat
(previous_ids[i], new_suffix, NULL);
              if (!g_hash_table_contains (seen, previous_desktop))
                {
                  g_hash_table_insert (seen, previous_desktop,
previous_desktop);
                  g_ptr_array_add (merged, g_steal_pointer
(&previous_desktop));
                }
            }

          if (merged->len > 0)
            {
              g_ptr_array_add (merged, NULL);
              g_key_file_set_string_list (keyfile,
                                          G_KEY_FILE_DESKTOP_GROUP,
                                          X_FLATPAK_RENAMED_FROM,
                                          (const char * const *) merged-
>pdata,
                                          merged->len - 1);
            }
        }
    }

  groups = g_key_file_get_groups (keyfile, NULL);

  for (i = 0; groups[i] != NULL; i++)
    {
      g_auto(GStrv) flatpak_run_opts = g_key_file_get_string_list
(keyfile, groups[i], "X-Flatpak-RunOptions", NULL, NULL);
      g_autofree char *flatpak_run_args =
format_flatpak_run_args_from_run_opts (flatpak_run_opts);

      g_key_file_remove_key (keyfile, groups[i], "X-Flatpak-RunOptions",
NULL);
      g_key_file_remove_key (keyfile, groups[i], "TryExec", NULL);

      /* Remove this to make sure nothing tries to execute it outside the
sandbox*/
      g_key_file_remove_key (keyfile, groups[i], "X-GNOME-Bugzilla-
ExtraInfoScript", NULL);

      new_exec = g_string_new ("");
      g_string_append_printf (new_exec,
                              FLATPAK_BINDIR "/flatpak run --branch=%s --
arch=%s",
                              escaped_branch,
```

```
                                escaped_arch);

      if (flatpak_run_args != NULL)
        g_string_append_printf (new_exec, "%s", flatpak_run_args);

      old_exec = g_key_file_get_string (keyfile, groups[i], "Exec",
NULL);
      if (old_exec && g_shell_parse_argv (old_exec, &old_argc, &old_argv,
NULL) && old_argc >= 1)
        {
          int j;
          g_autofree char *command = maybe_quote (old_argv[0]);

          g_string_append_printf (new_exec, " --command=%s", command);

          for (j = 1; j < old_argc; j++)
            {
              if (strcasecmp (old_argv[j], "%f") == 0 ||
                  strcasecmp (old_argv[j], "%u") == 0)
                {
                  g_string_append (new_exec, " --file-forwarding");
                  break;
                }
            }

          g_string_append (new_exec, " ");
          g_string_append (new_exec, escaped_app);

          for (j = 1; j < old_argc; j++)
            {
              g_autofree char *arg = maybe_quote (old_argv[j]);

              if (strcasecmp (arg, "%f") == 0)
                g_string_append_printf (new_exec, " @@ %s @@", arg);
              else if (strcasecmp (arg, "%u") == 0)
                g_string_append_printf (new_exec, " @@u %s @@", arg);
              else if (strcmp (arg, "@@") == 0 || strcmp (arg, "@@u") ==
0)
                g_print (_("Skipping invalid Exec argument %s\n"), arg);
              else
                g_string_append_printf (new_exec, " %s", arg);
            }
        }
      else
        {
          g_string_append (new_exec, " ");
          g_string_append (new_exec, escaped_app);
        }

      g_key_file_set_string (keyfile, groups[i],
G_KEY_FILE_DESKTOP_KEY_EXEC, new_exec->str);
    }

  new_data = g_key_file_to_data (keyfile, &new_data_len, error);
```

```
  if (new_data == NULL)
    goto out;

  if (!flatpak_open_in_tmpdir_at (parent_fd, 0755, tmpfile_name,
&out_stream, cancellable, error))
    goto out;

  if (!g_output_stream_write_all (out_stream, new_data, new_data_len,
NULL, cancellable, error))
    goto out;

  if (!g_output_stream_close (out_stream, cancellable, error))
    goto out;

  if (target)
    *target = g_steal_pointer (&tmpfile_name);

  ret = TRUE;
out:

  if (new_exec != NULL)
    g_string_free (new_exec, TRUE);

  return ret;
}
<sep>
void resolveOrPushdowns(MatchExpression* tree) {
    if (tree->numChildren() == 0) {
        return;
    }
    if (MatchExpression::AND == tree->matchType()) {
        AndMatchExpression* andNode =
static_cast<AndMatchExpression*>(tree);
        MatchExpression* indexedOr = getIndexedOr(andNode);

        for (size_t i = 0; i < andNode->numChildren(); ++i) {
            auto child = andNode->getChild(i);
            if (child->getTag() && child->getTag()->getType() ==
TagType::OrPushdownTag) {
                invariant(indexedOr);
                OrPushdownTag* orPushdownTag =
static_cast<OrPushdownTag*>(child->getTag());
                auto destinations = orPushdownTag->releaseDestinations();
                auto indexTag = orPushdownTag->releaseIndexTag();
                child->setTag(nullptr);
                if (pushdownNode(child, indexedOr,
std::move(destinations)) && !indexTag) {

                    // indexedOr can completely satisfy the predicate
specified in child, so we can
                    // trim it. We could remove the child even if it had
an index tag for this
                    // position, but that could make the index tagging of
the tree wrong.
```

```
                    auto ownedChild = andNode->removeChild(i);

                    // We removed child i, so decrement the child index.
                    --i;
                } else {
                    child->setTag(indexTag.release());
                }
            } else if (child->matchType() == MatchExpression::NOT &&
child->getChild(0)->getTag() &&
                       child->getChild(0)->getTag()->getType() ==
TagType::OrPushdownTag) {
                invariant(indexedOr);
                OrPushdownTag* orPushdownTag =
                    static_cast<OrPushdownTag*>(child->getChild(0)-
>getTag());
                auto destinations = orPushdownTag->releaseDestinations();
                auto indexTag = orPushdownTag->releaseIndexTag();
                child->getChild(0)->setTag(nullptr);

                // Push down the NOT and its child.
                if (pushdownNode(child, indexedOr,
std::move(destinations)) && !indexTag) {

                    // indexedOr can completely satisfy the predicate
specified in child, so we can
                    // trim it. We could remove the child even if it had
an index tag for this
                    // position, but that could make the index tagging of
the tree wrong.
                    auto ownedChild = andNode->removeChild(i);

                    // We removed child i, so decrement the child index.
                    --i;
                } else {
                    child->getChild(0)->setTag(indexTag.release());
                }
            } else if (child->matchType() ==
MatchExpression::ELEM_MATCH_OBJECT) {

                // Push down all descendants of child with
OrPushdownTags.
                std::vector<MatchExpression*> orPushdownDescendants;
                getElemMatchOrPushdownDescendants(child,
&orPushdownDescendants);
                if (!orPushdownDescendants.empty()) {
                    invariant(indexedOr);
                }
                for (auto descendant : orPushdownDescendants) {
                    OrPushdownTag* orPushdownTag =
                        static_cast<OrPushdownTag*>(descendant-
>getTag());
                    auto destinations = orPushdownTag-
>releaseDestinations();
                    auto indexTag = orPushdownTag->releaseIndexTag();
```

```
                    descendant->setTag(nullptr);
                    pushdownNode(descendant, indexedOr,
std::move(destinations));
                    descendant->setTag(indexTag.release());

                    // We cannot trim descendants of an $elemMatch
object, since the filter must
                    // be applied in its entirety.
                }
            }
        }
    }
    for (size_t i = 0; i < tree->numChildren(); ++i) {
        resolveOrPushdowns(tree->getChild(i));
    }
}
<sep>
date_s_httpdate(int argc, VALUE *argv, VALUE klass)
{
    VALUE str, sg;

    rb_scan_args(argc, argv, "02", &str, &sg);

    switch (argc) {
      case 0:
      str = rb_str_new2("Mon, 01 Jan -4712 00:00:00 GMT");
      case 1:
      sg = INT2FIX(DEFAULT_SG);
    }

    {
      VALUE hash = date_s__httpdate(klass, str);
      return d_new_by_frags(klass, hash, sg);
    }
}
<sep>
compute_O_value(std::string const& user_password,
           std::string const& owner_password,
           QPDF::EncryptionData const& data)
{
    // Algorithm 3.3 from the PDF 1.7 Reference Manual

    unsigned char O_key[OU_key_bytes_V4];
    compute_O_rc4_key(user_password, owner_password, data, O_key);

    char upass[key_bytes];
    pad_or_truncate_password_V4(user_password, upass);
    std::string k1(reinterpret_cast<char*>(O_key), OU_key_bytes_V4);
    pad_short_parameter(k1, data.getLengthBytes());
    iterate_rc4(QUtil::unsigned_char_pointer(upass), key_bytes,
           O_key, data.getLengthBytes(),
                (data.getR() >= 3) ? 20 : 1, false);
    return std::string(upass, key_bytes);
}
```

```
<sep>
void AES::encrypt(const byte* inBlock, const byte* xorBlock,
                  byte* outBlock) const
{
    word32 s0, s1, s2, s3;
    word32 t0, t1, t2, t3;

    const word32 *rk = key_;
    /*
     * map byte array block to cipher state
     * and add initial round key:
     */
    gpBlock::Get(inBlock)(s0)(s1)(s2)(s3);
    s0 ^= rk[0];
    s1 ^= rk[1];
    s2 ^= rk[2];
    s3 ^= rk[3];

    /*
     * Nr - 1 full rounds:
     */

    unsigned int r = rounds_ >> 1;
    for (;;) {
        t0 =
            Te0[GETBYTE(s0, 3)] ^
            Te1[GETBYTE(s1, 2)]  ^
            Te2[GETBYTE(s2, 1)]  ^
            Te3[GETBYTE(s3, 0)]  ^
            rk[4];
        t1 =
            Te0[GETBYTE(s1, 3)] ^
            Te1[GETBYTE(s2, 2)]  ^
            Te2[GETBYTE(s3, 1)]  ^
            Te3[GETBYTE(s0, 0)]  ^
            rk[5];
        t2 =
            Te0[GETBYTE(s2, 3)] ^
            Te1[GETBYTE(s3, 2)]  ^
            Te2[GETBYTE(s0, 1)]  ^
            Te3[GETBYTE(s1, 0)]  ^
            rk[6];
        t3 =
            Te0[GETBYTE(s3, 3)] ^
            Te1[GETBYTE(s0, 2)]  ^
            Te2[GETBYTE(s1, 1)]  ^
            Te3[GETBYTE(s2, 0)]  ^
            rk[7];

        rk += 8;
        if (--r == 0) {
            break;
        }
```

```
    s0 =
        Te0[GETBYTE(t0, 3)] ^
        Te1[GETBYTE(t1, 2)] ^
        Te2[GETBYTE(t2, 1)] ^
        Te3[GETBYTE(t3, 0)] ^
        rk[0];
    s1 =
        Te0[GETBYTE(t1, 3)] ^
        Te1[GETBYTE(t2, 2)] ^
        Te2[GETBYTE(t3, 1)] ^
        Te3[GETBYTE(t0, 0)] ^
        rk[1];
    s2 =
        Te0[GETBYTE(t2, 3)] ^
        Te1[GETBYTE(t3, 2)] ^
        Te2[GETBYTE(t0, 1)] ^
        Te3[GETBYTE(t1, 0)] ^
        rk[2];
    s3 =
        Te0[GETBYTE(t3, 3)] ^
        Te1[GETBYTE(t0, 2)] ^
        Te2[GETBYTE(t1, 1)] ^
        Te3[GETBYTE(t2, 0)] ^
        rk[3];
}

/*
 * apply last round and
 * map cipher state to byte array block:
 */

s0 =
    (Te4[GETBYTE(t0, 3)] & 0xff000000) ^
    (Te4[GETBYTE(t1, 2)] & 0x00ff0000) ^
    (Te4[GETBYTE(t2, 1)] & 0x0000ff00) ^
    (Te4[GETBYTE(t3, 0)] & 0x000000ff) ^
    rk[0];
s1 =
    (Te4[GETBYTE(t1, 3)] & 0xff000000) ^
    (Te4[GETBYTE(t2, 2)] & 0x00ff0000) ^
    (Te4[GETBYTE(t3, 1)] & 0x0000ff00) ^
    (Te4[GETBYTE(t0, 0)] & 0x000000ff) ^
    rk[1];
s2 =
    (Te4[GETBYTE(t2, 3)] & 0xff000000) ^
    (Te4[GETBYTE(t3, 2)] & 0x00ff0000) ^
    (Te4[GETBYTE(t0, 1)] & 0x0000ff00) ^
    (Te4[GETBYTE(t1, 0)] & 0x000000ff) ^
    rk[2];
s3 =
    (Te4[GETBYTE(t3, 3)] & 0xff000000) ^
    (Te4[GETBYTE(t0, 2)] & 0x00ff0000) ^
    (Te4[GETBYTE(t1, 1)] & 0x0000ff00) ^
    (Te4[GETBYTE(t2, 0)] & 0x000000ff) ^
```

```
        rk[3];


    gpBlock::Put(xorBlock, outBlock)(s0)(s1)(s2)(s3);
}
<sep>
int tls1_alert_code(int code)
      {
      switch (code)
            {
      case SSL_AD_CLOSE_NOTIFY:    return(SSL3_AD_CLOSE_NOTIFY);
      case SSL_AD_UNEXPECTED_MESSAGE:
      return(SSL3_AD_UNEXPECTED_MESSAGE);
      case SSL_AD_BAD_RECORD_MAC:  return(SSL3_AD_BAD_RECORD_MAC);
      case SSL_AD_DECRYPTION_FAILED:
      return(TLS1_AD_DECRYPTION_FAILED);
      case SSL_AD_RECORD_OVERFLOW: return(TLS1_AD_RECORD_OVERFLOW);
      case
SSL_AD_DECOMPRESSION_FAILURE:return(SSL3_AD_DECOMPRESSION_FAILURE);
      case SSL_AD_HANDSHAKE_FAILURE:
      return(SSL3_AD_HANDSHAKE_FAILURE);
      case SSL_AD_NO_CERTIFICATE:  return(-1);
      case SSL_AD_BAD_CERTIFICATE: return(SSL3_AD_BAD_CERTIFICATE);
      case
SSL_AD_UNSUPPORTED_CERTIFICATE:return(SSL3_AD_UNSUPPORTED_CERTIFICATE);
      case
SSL_AD_CERTIFICATE_REVOKED:return(SSL3_AD_CERTIFICATE_REVOKED);
      case
SSL_AD_CERTIFICATE_EXPIRED:return(SSL3_AD_CERTIFICATE_EXPIRED);
      case
SSL_AD_CERTIFICATE_UNKNOWN:return(SSL3_AD_CERTIFICATE_UNKNOWN);
      case SSL_AD_ILLEGAL_PARAMETER:
      return(SSL3_AD_ILLEGAL_PARAMETER);
      case SSL_AD_UNKNOWN_CA:           return(TLS1_AD_UNKNOWN_CA);
      case SSL_AD_ACCESS_DENIED:   return(TLS1_AD_ACCESS_DENIED);
      case SSL_AD_DECODE_ERROR:    return(TLS1_AD_DECODE_ERROR);
      case SSL_AD_DECRYPT_ERROR:   return(TLS1_AD_DECRYPT_ERROR);
      case SSL_AD_EXPORT_RESTRICTION:
      return(TLS1_AD_EXPORT_RESTRICTION);
      case SSL_AD_PROTOCOL_VERSION:     return(TLS1_AD_PROTOCOL_VERSION);
      case
SSL_AD_INSUFFICIENT_SECURITY:return(TLS1_AD_INSUFFICIENT_SECURITY);
      case SSL_AD_INTERNAL_ERROR:  return(TLS1_AD_INTERNAL_ERROR);
      case SSL_AD_USER_CANCELLED:  return(TLS1_AD_USER_CANCELLED);
      case SSL_AD_NO_RENEGOTIATION:     return(TLS1_AD_NO_RENEGOTIATION);
      case SSL_AD_UNSUPPORTED_EXTENSION:
return(TLS1_AD_UNSUPPORTED_EXTENSION);
      case SSL_AD_CERTIFICATE_UNOBTAINABLE:
return(TLS1_AD_CERTIFICATE_UNOBTAINABLE);
      case SSL_AD_UNRECOGNIZED_NAME:
      return(TLS1_AD_UNRECOGNIZED_NAME);
      case SSL_AD_BAD_CERTIFICATE_STATUS_RESPONSE:
return(TLS1_AD_BAD_CERTIFICATE_STATUS_RESPONSE);
```

```
      case SSL_AD_BAD_CERTIFICATE_HASH_VALUE:
return(TLS1_AD_BAD_CERTIFICATE_HASH_VALUE);
      case
SSL_AD_UNKNOWN_PSK_IDENTITY:return(TLS1_AD_UNKNOWN_PSK_IDENTITY);
#if 0 /* not appropriate for TLS, not used for DTLS */
      case DTLS1_AD_MISSING_HANDSHAKE_MESSAGE: return
                                (DTLS1_AD_MISSING_HANDSHAKE_MESSAGE);
#endif
      default:               return(-1);
            }
      }
<sep>
get_uncompressed_data(struct archive_read *a, const void **buff, size_t
size,
    size_t minimum)
{
      struct _7zip *zip = (struct _7zip *)a->format->data;
      ssize_t bytes_avail;

      if (zip->codec == _7Z_COPY && zip->codec2 == (unsigned long)-1) {
            /* Copy mode. */

            /*
             * Note: '1' here is a performance optimization.
             * Recall that the decompression layer returns a count of
             * available bytes; asking for more than that forces the
             * decompressor to combine reads by copying data.
             */
            *buff = __archive_read_ahead(a, 1, &bytes_avail);
            if (bytes_avail <= 0) {
                  archive_set_error(&a->archive,
                      ARCHIVE_ERRNO_FILE_FORMAT,
                      "Truncated 7-Zip file data");
                  return (ARCHIVE_FATAL);
            }
            if ((size_t)bytes_avail >
                zip->uncompressed_buffer_bytes_remaining)
                  bytes_avail = (ssize_t)
                      zip->uncompressed_buffer_bytes_remaining;
            if ((size_t)bytes_avail > size)
                  bytes_avail = (ssize_t)size;

            zip->pack_stream_bytes_unconsumed = bytes_avail;
      } else if (zip->uncompressed_buffer_pointer == NULL) {
            /* Decompression has failed. */
            archive_set_error(&(a->archive),
                ARCHIVE_ERRNO_MISC, "Damaged 7-Zip archive");
            return (ARCHIVE_FATAL);
      } else {
            /* Packed mode. */
            if (minimum > zip->uncompressed_buffer_bytes_remaining) {
                  /*
                   * If remaining uncompressed data size is less than
                   * the minimum size, fill the buffer up to the
```

```
                 * minimum size.
                 */
                if (extract_pack_stream(a, minimum) < 0)
                     return (ARCHIVE_FATAL);
            }
            if (size > zip->uncompressed_buffer_bytes_remaining)
                 bytes_avail = (ssize_t)
                     zip->uncompressed_buffer_bytes_remaining;
            else
                 bytes_avail = (ssize_t)size;
            *buff = zip->uncompressed_buffer_pointer;
            zip->uncompressed_buffer_pointer += bytes_avail;
      }
      zip->uncompressed_buffer_bytes_remaining -= bytes_avail;
      return (bytes_avail);
}
<sep>
_libssh2_channel_flush(LIBSSH2_CHANNEL *channel, int streamid)
{
    if(channel->flush_state == libssh2_NB_state_idle) {
        LIBSSH2_PACKET *packet =
            _libssh2_list_first(&channel->session->packets);
        channel->flush_refund_bytes = 0;
        channel->flush_flush_bytes = 0;

        while(packet) {
            LIBSSH2_PACKET *next = _libssh2_list_next(&packet->node);
            unsigned char packet_type = packet->data[0];

            if(((packet_type == SSH_MSG_CHANNEL_DATA)
                 || (packet_type == SSH_MSG_CHANNEL_EXTENDED_DATA))
                && (_libssh2_ntohu32(packet->data + 1) == channel-
>local.id)) {
                /* It's our channel at least */
                long packet_stream_id =
                    (packet_type == SSH_MSG_CHANNEL_DATA) ? 0 :
                    _libssh2_ntohu32(packet->data + 5);
                if((streamid == LIBSSH2_CHANNEL_FLUSH_ALL)
                    || ((packet_type == SSH_MSG_CHANNEL_EXTENDED_DATA)
                        && ((streamid ==
LIBSSH2_CHANNEL_FLUSH_EXTENDED_DATA)
                            || (streamid == packet_stream_id)))
                    || ((packet_type == SSH_MSG_CHANNEL_DATA)
                        && (streamid == 0))) {
                    int bytes_to_flush = packet->data_len - packet-
>data_head;

                    _libssh2_debug(channel->session, LIBSSH2_TRACE_CONN,
                                   "Flushing %d bytes of data from stream
"
                                   "%lu on channel %lu/%lu",
                                   bytes_to_flush, packet_stream_id,
                                   channel->local.id, channel-
>remote.id);
```

```
                    /* It's one of the streams we wanted to flush */
                    channel->flush_refund_bytes += packet->data_len - 13;
                    channel->flush_flush_bytes += bytes_to_flush;

                    LIBSSH2_FREE(channel->session, packet->data);

                    /* remove this packet from the parent's list */
                    _libssh2_list_remove(&packet->node);
                    LIBSSH2_FREE(channel->session, packet);
                }
            }
            packet = next;
        }

        channel->flush_state = libssh2_NB_state_created;
    }

    channel->read_avail -= channel->flush_flush_bytes;
    channel->remote.window_size -= channel->flush_flush_bytes;

    if(channel->flush_refund_bytes) {
        int rc;

        rc = _libssh2_channel_receive_window_adjust(channel,
                                                    channel-
>flush_refund_bytes,
                                                    1, NULL);
        if(rc == LIBSSH2_ERROR_EAGAIN)
            return rc;
    }

    channel->flush_state = libssh2_NB_state_idle;

    return channel->flush_flush_bytes;
}
<sep>
static void gf_m2ts_process_pat(GF_M2TS_Demuxer *ts, GF_M2TS_SECTION_ES
*ses, GF_List *sections, u8 table_id, u16 ex_table_id, u8 version_number,
u8 last_section_number, u32 status)
{
      GF_M2TS_Program *prog;
      GF_M2TS_SECTION_ES *pmt;
      u32 i, nb_progs, evt_type;
      u32 nb_sections;
      u32 data_size;
      unsigned char *data;
      GF_M2TS_Section *section;

      /*wait for the last section */
      if (!(status&GF_M2TS_TABLE_END)) return;

      /*skip if already received*/
      if (status&GF_M2TS_TABLE_REPEAT) {
```

```
            if (ts->on_event) ts->on_event(ts, GF_M2TS_EVT_PAT_REPEAT,
NULL);
            return;
      }

      nb_sections = gf_list_count(sections);
      if (nb_sections > 1) {
            GF_LOG(GF_LOG_WARNING, GF_LOG_CONTAINER, ("PAT on multiple
sections not supported\n"));
      }

      section = (GF_M2TS_Section *)gf_list_get(sections, 0);
      data = section->data;
      data_size = section->data_size;

      if (!(status&GF_M2TS_TABLE_UPDATE) && gf_list_count(ts->programs))
{
            if (ts->pat->demux_restarted) {
                  ts->pat->demux_restarted = 0;
            } else {
                  GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER, ("Multiple
different PAT on single TS found, ignoring new PAT declaration (table id
%d - extended table id %d)\n", table_id, ex_table_id));
            }
            return;
      }
      nb_progs = data_size / 4;

      for (i=0; i<nb_progs; i++) {
            u16 number, pid;
            number = (data[0]<<8) | data[1];
            pid = (data[2]&0x1f)<<8 | data[3];
            data += 4;
            if (number==0) {
                  if (!ts->nit) {
                        ts->nit =
gf_m2ts_section_filter_new(gf_m2ts_process_nit, 0);
                  }
            } else {
                  GF_SAFEALLOC(prog, GF_M2TS_Program);
                  if (!prog) {
                        GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER, ("Fail to
allocate program for pid %d\n", pid));
                        return;
                  }
                  prog->streams = gf_list_new();
                  prog->pmt_pid = pid;
                  prog->number = number;
                  prog->ts = ts;
                  gf_list_add(ts->programs, prog);
                  GF_SAFEALLOC(pmt, GF_M2TS_SECTION_ES);
                  if (!pmt) {
                        GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER, ("Fail to
allocate pmt filter for pid %d\n", pid));
```

```
                        return;
                    }
                    pmt->flags = GF_M2TS_ES_IS_SECTION;
                    gf_list_add(prog->streams, pmt);
                    pmt->pid = prog->pmt_pid;
                    pmt->program = prog;
                    ts->ess[pmt->pid] = (GF_M2TS_ES *)pmt;
                    pmt->sec =
gf_m2ts_section_filter_new(gf_m2ts_process_pmt, 0);
                }
        }

        evt_type = (status&GF_M2TS_TABLE_UPDATE) ? GF_M2TS_EVT_PAT_UPDATE :
GF_M2TS_EVT_PAT_FOUND;
        if (ts->on_event) ts->on_event(ts, evt_type, NULL);
}
<sep>
SPL_METHOD(SplObjectStorage, getHash)
{
        zval *obj;
        char *hash;

        if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "o", &obj) ==
FAILURE) {
                return;
        }

        hash = emalloc(33);
        php_spl_object_hash(obj, hash TSRMLS_CC);

        RETVAL_STRING(hash, 0);

} /* }}} */
<sep>
int wolfSSH_SFTP_RecvRead(WOLFSSH* ssh, int reqId, byte* data, word32
maxSz)
#ifndef USE_WINDOWS_API
{
    WFD    fd;
    word32 sz;
    int    ret;
    word32 idx  = 0;
    word32 ofst[2] = {0, 0};

    byte*  out;
    word32 outSz;

    char* res  = NULL;
    char err[] = "Read File Error";
    char eof[] = "Read EOF";
    byte type = WOLFSSH_FTP_FAILURE;

    if (ssh == NULL) {
        return WS_BAD_ARGUMENT;
```

```
    }

    WLOG(WS_LOG_SFTP, "Receiving WOLFSSH_FTP_READ");

    /* get file handle */
    ato32(data + idx, &sz); idx += UINT32_SZ;
    if (sz + idx > maxSz || sz > WOLFSSH_MAX_HANDLE) {
        return WS_BUFFER_E;
    }
    WMEMSET((byte*)&fd, 0, sizeof(WFD));
    WMEMCPY((byte*)&fd, data + idx, sz); idx += sz;

    /* get offset into file */
    ato32(data + idx, &ofst[1]); idx += UINT32_SZ;
    ato32(data + idx, &ofst[0]); idx += UINT32_SZ;

    /* get length to be read */
    ato32(data + idx, &sz);

    /* read from handle and send data back to client */
    out = (byte*)WMALLOC(sz + WOLFSSH_SFTP_HEADER + UINT32_SZ,
            ssh->ctx->heap, DYNTYPE_BUFFER);
    if (out == NULL) {
        return WS_MEMORY_E;
    }

    ret = WPREAD(fd, out + UINT32_SZ + WOLFSSH_SFTP_HEADER, sz, ofst);
    if (ret < 0 || (word32)ret > sz) {
        WLOG(WS_LOG_SFTP, "Error reading from file");
        res  = err;
        type = WOLFSSH_FTP_FAILURE;
        ret  = WS_BAD_FILE_E;
    }
    else {
        outSz = (word32)ret + WOLFSSH_SFTP_HEADER + UINT32_SZ;
    }

    /* eof */
    if (ret == 0) {
        WLOG(WS_LOG_SFTP, "Error reading from file, EOF");
        res = eof;
        type = WOLFSSH_FTP_EOF;
        ret = WS_SUCCESS; /* end of file is not fatal error */
    }

    if (res != NULL) {
        if (wolfSSH_SFTP_CreateStatus(ssh, type, reqId, res, "English",
NULL,
                &outSz) != WS_SIZE_ONLY) {
            WFREE(out, ssh->ctx->heap, DYNTYPE_BUFFER);
            return WS_FATAL_ERROR;
        }
        if (outSz > sz) {
            /* need to increase buffer size for holding status packet */
```

```
            WFREE(out, ssh->ctx->heap, DYNTYPE_BUFFER);
            out = (byte*)WMALLOC(outSz, ssh->ctx->heap, DYNTYPE_BUFFER);
            if (out == NULL) {
                return WS_MEMORY_E;
            }
        }
        if (wolfSSH_SFTP_CreateStatus(ssh, type, reqId, res, "English",
out,
                    &outSz) != WS_SUCCESS) {
            WFREE(out, ssh->ctx->heap, DYNTYPE_BUFFER);
            return WS_FATAL_ERROR;
        }
    }
    else {
        SFTP_CreatePacket(ssh, WOLFSSH_FTP_DATA, out, outSz, NULL, 0);
    }

    /* set send out buffer, "out" is taken by ssh  */
    wolfSSH_SFTP_RecvSetSend(ssh, out, outSz);
    return ret;
}
<sep>
state_separate_contexts (position_set const *s)
{
  int separate_contexts = 0;
  unsigned int j;

  for (j = 0; j < s->nelem; ++j)
    {
      if (PREV_NEWLINE_DEPENDENT (s->elems[j].constraint))
        separate_contexts |= CTX_NEWLINE;
      if (PREV_LETTER_DEPENDENT (s->elems[j].constraint))
        separate_contexts |= CTX_LETTER;
    }

  return separate_contexts;
}
<sep>
utf32be_mbc_to_code(const UChar* p, const UChar* end ARG_UNUSED)
{
  return (OnigCodePoint )(((p[0] * 256 + p[1]) * 256 + p[2]) * 256 +
p[3]);
}
<sep>
void jpc_qmfb_join_col(jpc_fix_t *a, int numrows, int stride,
  int parity)
{

      int bufsize = JPC_CEILDIVPOW2(numrows, 1);
      jpc_fix_t joinbuf[QMFB_JOINBUFSIZE];
      jpc_fix_t *buf = joinbuf;
      register jpc_fix_t *srcptr;
      register jpc_fix_t *dstptr;
      register int n;
```

```
      int hstartcol;

      /* Allocate memory for the join buffer from the heap. */
      if (bufsize > QMFB_JOINBUFSIZE) {
            if (!(buf = jas_malloc(bufsize * sizeof(jpc_fix_t)))) {
                  /* We have no choice but to commit suicide. */
                  abort();
            }
      }

      hstartcol = (numrows + 1 - parity) >> 1;

      /* Save the samples from the lowpass channel. */
      n = hstartcol;
      srcptr = &a[0];
      dstptr = buf;
      while (n-- > 0) {
            *dstptr = *srcptr;
            srcptr += stride;
            ++dstptr;
      }
      /* Copy the samples from the highpass channel into place. */
      srcptr = &a[hstartcol * stride];
      dstptr = &a[(1 - parity) * stride];
      n = numrows - hstartcol;
      while (n-- > 0) {
            *dstptr = *srcptr;
            dstptr += 2 * stride;
            srcptr += stride;
      }
      /* Copy the samples from the lowpass channel into place. */
      srcptr = buf;
      dstptr = &a[parity * stride];
      n = hstartcol;
      while (n-- > 0) {
            *dstptr = *srcptr;
            dstptr += 2 * stride;
            ++srcptr;
      }

      /* If the join buffer was allocated on the heap, free this memory.
*/
      if (buf != joinbuf) {
            jas_free(buf);
      }

}
<sep>
compat_mpt_command(struct file *filp, unsigned int cmd,
                   unsigned long arg)
{
      struct mpt_ioctl_command32 karg32;
      struct mpt_ioctl_command32 __user *uarg = (struct
mpt_ioctl_command32 __user *) arg;
```

```
      struct mpt_ioctl_command karg;
      MPT_ADAPTER *iocp = NULL;
      int iocnum, iocnumX;
      int nonblock = (filp->f_flags & O_NONBLOCK);
      int ret;

      if (copy_from_user(&karg32, (char __user *)arg, sizeof(karg32)))
            return -EFAULT;

      /* Verify intended MPT adapter */
      iocnumX = karg32.hdr.iocnum & 0xFF;
      if (((iocnum = mpt_verify_adapter(iocnumX, &iocp)) < 0) ||
          (iocp == NULL)) {
            printk(KERN_DEBUG MYNAM "::compat_mpt_command @%d - ioc%d not
found!\n",
                  __LINE__, iocnumX);
            return -ENODEV;
      }

      if ((ret = mptctl_syscall_down(iocp, nonblock)) != 0)
            return ret;

      dctlprintk(iocp, printk(MYIOC_s_DEBUG_FMT "compat_mpt_command()
called\n",
          iocp->name));
      /* Copy data to karg */
      karg.hdr.iocnum = karg32.hdr.iocnum;
      karg.hdr.port = karg32.hdr.port;
      karg.timeout = karg32.timeout;
      karg.maxReplyBytes = karg32.maxReplyBytes;

      karg.dataInSize = karg32.dataInSize;
      karg.dataOutSize = karg32.dataOutSize;
      karg.maxSenseBytes = karg32.maxSenseBytes;
      karg.dataSgeOffset = karg32.dataSgeOffset;

      karg.replyFrameBufPtr = (char __user *)(unsigned
long)karg32.replyFrameBufPtr;
      karg.dataInBufPtr = (char __user *)(unsigned
long)karg32.dataInBufPtr;
      karg.dataOutBufPtr = (char __user *)(unsigned
long)karg32.dataOutBufPtr;
      karg.senseDataPtr = (char __user *)(unsigned
long)karg32.senseDataPtr;

      /* Pass new structure to do_mpt_command
       */
      ret = mptctl_do_mpt_command (karg, &uarg->MF);

      mutex_unlock(&iocp->ioctl_cmds.mutex);

      return ret;
}
<sep>
```

```
gdImagePtr gdImageCreateFromGifCtx(gdIOCtxPtr fd) /* {{{ */
{
      int BitPixel;
#if 0
      int ColorResolution;
      int Background;
      int AspectRatio;
#endif
      int Transparent = (-1);
      unsigned char   buf[16];
      unsigned char   c;
      unsigned char   ColorMap[3][MAXCOLORMAPSIZE];
      unsigned char   localColorMap[3][MAXCOLORMAPSIZE];
      int             imw, imh, screen_width, screen_height;
      int             gif87a, useGlobalColormap;
      int             bitPixel;
      int          i;
      /*1.4//int             imageCount = 0; */

      int ZeroDataBlock = FALSE;
      int haveGlobalColormap;
      gdImagePtr im = 0;

      /*1.4//imageNumber = 1; */
      if (! ReadOK(fd,buf,6)) {
            return 0;
      }
      if (strncmp((char *)buf,"GIF",3) != 0) {
            return 0;
      }

      if (memcmp((char *)buf+3, "87a", 3) == 0) {
            gif87a = 1;
      } else if (memcmp((char *)buf+3, "89a", 3) == 0) {
            gif87a = 0;
      } else {
            return 0;
      }

      if (! ReadOK(fd,buf,7)) {
            return 0;
      }

      BitPixel        = 2<<(buf[4]&0x07);
#if 0
      ColorResolution = (int) (((buf[4]&0x70)>>3)+1);
      Background      = buf[5];
      AspectRatio     = buf[6];
#endif
      screen_width = imw = LM_to_uint(buf[0],buf[1]);
      screen_height = imh = LM_to_uint(buf[2],buf[3]);

      haveGlobalColormap = BitSet(buf[4], LOCALCOLORMAP);    /* Global
Colormap */
```

```
      if (haveGlobalColormap) {
            if (ReadColorMap(fd, BitPixel, ColorMap)) {
                  return 0;
            }
      }

      for (;;) {
            int top, left;
            int width, height;

            if (! ReadOK(fd,&c,1)) {
                  return 0;
            }
            if (c == ';') {         /* GIF terminator */
                  goto terminated;
            }

            if (c == '!') {         /* Extension */
                  if (! ReadOK(fd,&c,1)) {
                        return 0;
                  }
                  DoExtension(fd, c, &Transparent, &ZeroDataBlock);
                  continue;
            }

            if (c != ',') {         /* Not a valid start character */
                  continue;
            }

            /*1.4//++imageCount; */

            if (! ReadOK(fd,buf,9)) {
                  return 0;
            }

            useGlobalColormap = ! BitSet(buf[8], LOCALCOLORMAP);

            bitPixel = 1<<((buf[8]&0x07)+1);
            left = LM_to_uint(buf[0], buf[1]);
            top = LM_to_uint(buf[2], buf[3]);
            width = LM_to_uint(buf[4], buf[5]);
            height = LM_to_uint(buf[6], buf[7]);

            if (left + width > screen_width || top + height >
screen_height) {
                  if (VERBOSE) {
                        printf("Frame is not confined to screen
dimension.\n");
                  }
                  return 0;
            }

            if (!(im = gdImageCreate(width, height))) {
                  return 0;
```

```
            }
            im->interlace = BitSet(buf[8], INTERLACE);
            if (!useGlobalColormap) {
                  if (ReadColorMap(fd, bitPixel, localColorMap)) {
                        gdImageDestroy(im);
                        return 0;
                  }
                  ReadImage(im, fd, width, height, localColorMap,
                              BitSet(buf[8], INTERLACE), &ZeroDataBlock);
            } else {
                  if (!haveGlobalColormap) {
                        gdImageDestroy(im);
                        return 0;
                  }
                  ReadImage(im, fd, width, height,
                                    ColorMap,
                                    BitSet(buf[8], INTERLACE),
&ZeroDataBlock);
            }
            if (Transparent != (-1)) {
                  gdImageColorTransparent(im, Transparent);
            }
            goto terminated;
      }

terminated:
      /* Terminator before any image was declared! */
      if (!im) {
            return 0;
      }
      if (!im->colorsTotal) {
            gdImageDestroy(im);
            return 0;
      }
      /* Check for open colors at the end, so
         we can reduce colorsTotal and ultimately
         BitsPerPixel */
      for (i=((im->colorsTotal-1)); (i>=0); i--) {
            if (im->open[i]) {
                  im->colorsTotal--;
            } else {
                  break;
            }
      }
      return im;
}
<sep>
static inline struct futex_hash_bucket *queue_lock(struct futex_q *q)
{
      struct futex_hash_bucket *hb;

      get_futex_key_refs(&q->key);
      hb = hash_futex(&q->key);
      q->lock_ptr = &hb->lock;
```

```
	spin_lock(&hb->lock);
	return hb;
}
<sep>
CPH_METHOD(LoadFromFile)
{
	HRESULT res;
	char *filename, *fullpath;
	int filename_len;
	long flags = 0;
	OLECHAR *olefilename;
	CPH_FETCH();

	CPH_NO_OBJ();

	res = get_persist_file(helper);
	if (helper->ipf) {

		if (FAILURE == zend_parse_parameters(ZEND_NUM_ARGS()
TSRMLS_CC, "s|l",
					&filename, &filename_len, &flags)) {
			php_com_throw_exception(E_INVALIDARG, "Invalid
arguments" TSRMLS_CC);
			return;
		}

		if (!(fullpath = expand_filepath(filename, NULL TSRMLS_CC)))
{
			RETURN_FALSE;
		}

		if ((PG(safe_mode) && (!php_checkuid(fullpath, NULL,
CHECKUID_CHECK_FILE_AND_DIR))) ||
				php_check_open_basedir(fullpath TSRMLS_CC)) {
			efree(fullpath);
			RETURN_FALSE;
		}

		olefilename = php_com_string_to_olestring(fullpath,
strlen(fullpath), helper->codepage TSRMLS_CC);
		efree(fullpath);

		res = IPersistFile_Load(helper->ipf, olefilename, flags);
		efree(olefilename);

		if (FAILED(res)) {
			php_com_throw_exception(res, NULL TSRMLS_CC);
		}

	} else {
		php_com_throw_exception(res, NULL TSRMLS_CC);
	}
}
```

```
<sep>
ossl_asn1_decode0(unsigned char **pp, long length, long *offset, int
depth,
              int yield, long *num_read)
{
    unsigned char *start, *p;
    const unsigned char *p0;
    long len = 0, inner_read = 0, off = *offset, hlen;
    int tag, tc, j;
    VALUE asn1data, tag_class;

    p = *pp;
    start = p;
    p0 = p;
    j = ASN1_get_object(&p0, &len, &tag, &tc, length);
    p = (unsigned char *)p0;
    if(j & 0x80) ossl_raise(eASN1Error, NULL);
    if(len > length) ossl_raise(eASN1Error, "value is too short");
    if((tc & V_ASN1_PRIVATE) == V_ASN1_PRIVATE)
      tag_class = sym_PRIVATE;
    else if((tc & V_ASN1_CONTEXT_SPECIFIC) == V_ASN1_CONTEXT_SPECIFIC)
      tag_class = sym_CONTEXT_SPECIFIC;
    else if((tc & V_ASN1_APPLICATION) == V_ASN1_APPLICATION)
      tag_class = sym_APPLICATION;
    else
      tag_class = sym_UNIVERSAL;

    hlen = p - start;

    if(yield) {
      VALUE arg = rb_ary_new();
      rb_ary_push(arg, LONG2NUM(depth));
      rb_ary_push(arg, LONG2NUM(*offset));
      rb_ary_push(arg, LONG2NUM(hlen));
      rb_ary_push(arg, LONG2NUM(len));
      rb_ary_push(arg, (j & V_ASN1_CONSTRUCTED) ? Qtrue : Qfalse);
      rb_ary_push(arg, ossl_asn1_class2sym(tc));
      rb_ary_push(arg, INT2NUM(tag));
      rb_yield(arg);
    }

    if(j & V_ASN1_CONSTRUCTED) {
      *pp += hlen;
      off += hlen;
      asn1data = int_ossl_asn1_decode0_cons(pp, length, len, &off, depth,
yield, j, tag, tag_class, &inner_read);
      inner_read += hlen;
    }
    else {
      if ((j & 0x01) && (len == 0)) ossl_raise(eASN1Error, "Infinite
length for primitive value");
      asn1data = int_ossl_asn1_decode0_prim(pp, len, hlen, tag,
tag_class, &inner_read);
      off += hlen + len;
```

```
    }
    if (num_read)
      *num_read = inner_read;
    if (len != 0 && inner_read != hlen + len) {
      ossl_raise(eASN1Error,
                 "Type mismatch. Bytes read: %ld Bytes available: %ld",
                 inner_read, hlen + len);
    }

    *offset = off;
    return asn1data;
}
<sep>
static OPJ_BOOL opj_j2k_write_first_tile_part(opj_j2k_t *p_j2k,
        OPJ_BYTE * p_data,
        OPJ_UINT32 * p_data_written,
        OPJ_UINT32 p_total_data_size,
        opj_stream_private_t *p_stream,
        struct opj_event_mgr * p_manager)
{
    OPJ_UINT32 l_nb_bytes_written = 0;
    OPJ_UINT32 l_current_nb_bytes_written;
    OPJ_BYTE * l_begin_data = 00;

    opj_tcd_t * l_tcd = 00;
    opj_cp_t * l_cp = 00;

    l_tcd = p_j2k->m_tcd;
    l_cp = &(p_j2k->m_cp);

    l_tcd->cur_pino = 0;

    /*Get number of tile parts*/
    p_j2k->m_specific_param.m_encoder.m_current_poc_tile_part_number = 0;

    /* INDEX >> */
    /* << INDEX */

    l_current_nb_bytes_written = 0;
    l_begin_data = p_data;
    if (! opj_j2k_write_sot(p_j2k, p_data, &l_current_nb_bytes_written,
p_stream,
                            p_manager)) {
        return OPJ_FALSE;
    }

    l_nb_bytes_written += l_current_nb_bytes_written;
    p_data += l_current_nb_bytes_written;
    p_total_data_size -= l_current_nb_bytes_written;

    if (!OPJ_IS_CINEMA(l_cp->rsiz)) {
#if 0
        for (compno = 1; compno < p_j2k->m_private_image->numcomps;
compno++) {
```

```
            l_current_nb_bytes_written = 0;
            opj_j2k_write_coc_in_memory(p_j2k, compno, p_data,
&l_current_nb_bytes_written,
                                        p_manager);
            l_nb_bytes_written += l_current_nb_bytes_written;
            p_data += l_current_nb_bytes_written;
            p_total_data_size -= l_current_nb_bytes_written;

            l_current_nb_bytes_written = 0;
            opj_j2k_write_qcc_in_memory(p_j2k, compno, p_data,
&l_current_nb_bytes_written,
                                        p_manager);
            l_nb_bytes_written += l_current_nb_bytes_written;
            p_data += l_current_nb_bytes_written;
            p_total_data_size -= l_current_nb_bytes_written;
        }
#endif
        if (l_cp->tcps[p_j2k->m_current_tile_number].numpocs) {
            l_current_nb_bytes_written = 0;
            opj_j2k_write_poc_in_memory(p_j2k, p_data,
&l_current_nb_bytes_written,
                                        p_manager);
            l_nb_bytes_written += l_current_nb_bytes_written;
            p_data += l_current_nb_bytes_written;
            p_total_data_size -= l_current_nb_bytes_written;
        }
    }

    l_current_nb_bytes_written = 0;
    if (! opj_j2k_write_sod(p_j2k, l_tcd, p_data,
&l_current_nb_bytes_written,
                            p_total_data_size, p_stream, p_manager)) {
        return OPJ_FALSE;
    }

    l_nb_bytes_written += l_current_nb_bytes_written;
    * p_data_written = l_nb_bytes_written;

    /* Writing Psot in SOT marker */
    opj_write_bytes(l_begin_data + 6, l_nb_bytes_written,
                    4);                                     /* PSOT */

    if (OPJ_IS_CINEMA(l_cp->rsiz)) {
        opj_j2k_update_tlm(p_j2k, l_nb_bytes_written);
    }

    return OPJ_TRUE;
}
<sep>
void fput(struct file *file)
{
      if (atomic_long_dec_and_test(&file->f_count)) {
            struct task_struct *task = current;
```

```
            file_sb_list_del(file);
            if (likely(!in_interrupt() && !(task->flags & PF_KTHREAD))) {
                  init_task_work(&file->f_u.fu_rcuhead, ____fput);
                  if (!task_work_add(task, &file->f_u.fu_rcuhead, true))
                        return;
                  /*
                   * After this task has run exit_task_work(),
                   * task_work_add() will fail.  Fall through to delayed
                   * fput to avoid leaking *file.
                   */
            }

            if (llist_add(&file->f_u.fu_llist, &delayed_fput_list))
                  schedule_work(&delayed_fput_work);
      }
}
<sep>
static uint_fast32_t jpc_abstorelstepsize(jpc_fix_t absdelta, int
scaleexpn)
{
      int p;
      uint_fast32_t mant;
      uint_fast32_t expn;
      int n;

      if (absdelta < 0) {
            abort();
      }

      p = jpc_firstone(absdelta) - JPC_FIX_FRACBITS;
      n = 11 - jpc_firstone(absdelta);
      mant = ((n < 0) ? (absdelta >> (-n)) : (absdelta << n)) & 0x7ff;
      expn = scaleexpn - p;
      if (scaleexpn < p) {
            abort();
      }
      return JPC_QCX_EXPN(expn) | JPC_QCX_MANT(mant);
}
<sep>
flatpak_context_load_metadata (FlatpakContext *context,
                               GKeyFile       *metakey,
                               GError        **error)
{
  gboolean remove;
  g_auto(GStrv) groups = NULL;
  gsize i;

  if (g_key_file_has_key (metakey, FLATPAK_METADATA_GROUP_CONTEXT,
FLATPAK_METADATA_KEY_SHARED, NULL))
    {
      g_auto(GStrv) shares = g_key_file_get_string_list (metakey,
FLATPAK_METADATA_GROUP_CONTEXT,

FLATPAK_METADATA_KEY_SHARED, NULL, error);
```

```
      if (shares == NULL)
        return FALSE;

      for (i = 0; shares[i] != NULL; i++)
        {
          FlatpakContextShares share;

          share = flatpak_context_share_from_string (parse_negated
(shares[i], &remove), NULL);
          if (share == 0)
            g_debug ("Unknown share type %s", shares[i]);
          else
            {
              if (remove)
                flatpak_context_remove_shares (context, share);
              else
                flatpak_context_add_shares (context, share);
            }
        }
    }

  if (g_key_file_has_key (metakey, FLATPAK_METADATA_GROUP_CONTEXT,
FLATPAK_METADATA_KEY_SOCKETS, NULL))
    {
      g_auto(GStrv) sockets = g_key_file_get_string_list (metakey,
FLATPAK_METADATA_GROUP_CONTEXT,

FLATPAK_METADATA_KEY_SOCKETS, NULL, error);
      if (sockets == NULL)
        return FALSE;

      for (i = 0; sockets[i] != NULL; i++)
        {
          FlatpakContextSockets socket =
flatpak_context_socket_from_string (parse_negated (sockets[i], &remove),
NULL);
          if (socket == 0)
            g_debug ("Unknown socket type %s", sockets[i]);
          else
            {
              if (remove)
                flatpak_context_remove_sockets (context, socket);
              else
                flatpak_context_add_sockets (context, socket);
            }
        }
    }

  if (g_key_file_has_key (metakey, FLATPAK_METADATA_GROUP_CONTEXT,
FLATPAK_METADATA_KEY_DEVICES, NULL))
    {
      g_auto(GStrv) devices = g_key_file_get_string_list (metakey,
FLATPAK_METADATA_GROUP_CONTEXT,
```

```
FLATPAK_METADATA_KEY_DEVICES, NULL, error);
      if (devices == NULL)
        return FALSE;


      for (i = 0; devices[i] != NULL; i++)
        {
          FlatpakContextDevices device =
flatpak_context_device_from_string (parse_negated (devices[i], &remove),
NULL);
          if (device == 0)
            g_debug ("Unknown device type %s", devices[i]);
          else
            {
              if (remove)
                flatpak_context_remove_devices (context, device);
              else
                flatpak_context_add_devices (context, device);
            }
        }
    }

  if (g_key_file_has_key (metakey, FLATPAK_METADATA_GROUP_CONTEXT,
FLATPAK_METADATA_KEY_FEATURES, NULL))
    {
      g_auto(GStrv) features = g_key_file_get_string_list (metakey,
FLATPAK_METADATA_GROUP_CONTEXT,

FLATPAK_METADATA_KEY_FEATURES, NULL, error);
      if (features == NULL)
        return FALSE;


      for (i = 0; features[i] != NULL; i++)
        {
          FlatpakContextFeatures feature =
flatpak_context_feature_from_string (parse_negated (features[i],
&remove), NULL);
          if (feature == 0)
            g_debug ("Unknown feature type %s", features[i]);
          else
            {
              if (remove)
                flatpak_context_remove_features (context, feature);
              else
                flatpak_context_add_features (context, feature);
            }
        }
    }

  if (g_key_file_has_key (metakey, FLATPAK_METADATA_GROUP_CONTEXT,
FLATPAK_METADATA_KEY_FILESYSTEMS, NULL))
    {
```

```
      g_auto(GStrv) filesystems = g_key_file_get_string_list (metakey,
FLATPAK_METADATA_GROUP_CONTEXT,

FLATPAK_METADATA_KEY_FILESYSTEMS, NULL, error);
      if (filesystems == NULL)
        return FALSE;

      for (i = 0; filesystems[i] != NULL; i++)
        {
          const char *fs = parse_negated (filesystems[i], &remove);
          g_autofree char *filesystem = NULL;
          FlatpakFilesystemMode mode;

          if (!flatpak_context_parse_filesystem (fs, &filesystem, &mode,
NULL))
            g_debug ("Unknown filesystem type %s", filesystems[i]);
          else
            {
              if (remove)
                flatpak_context_take_filesystem (context, g_steal_pointer
(&filesystem),

FLATPAK_FILESYSTEM_MODE_NONE);
              else
                flatpak_context_take_filesystem (context, g_steal_pointer
(&filesystem), mode);
            }
        }
    }

  if (g_key_file_has_key (metakey, FLATPAK_METADATA_GROUP_CONTEXT,
FLATPAK_METADATA_KEY_PERSISTENT, NULL))
    {
      g_auto(GStrv) persistent = g_key_file_get_string_list (metakey,
FLATPAK_METADATA_GROUP_CONTEXT,

FLATPAK_METADATA_KEY_PERSISTENT, NULL, error);
      if (persistent == NULL)
        return FALSE;

      for (i = 0; persistent[i] != NULL; i++)
        flatpak_context_set_persistent (context, persistent[i]);
    }

  if (g_key_file_has_group (metakey,
FLATPAK_METADATA_GROUP_SESSION_BUS_POLICY))
    {
      g_auto(GStrv) keys = NULL;
      gsize keys_count;

      keys = g_key_file_get_keys (metakey,
FLATPAK_METADATA_GROUP_SESSION_BUS_POLICY, &keys_count, NULL);
      for (i = 0; i < keys_count; i++)
        {
```

```
          const char *key = keys[i];
          g_autofree char *value = g_key_file_get_string (metakey,
FLATPAK_METADATA_GROUP_SESSION_BUS_POLICY, key, NULL);
          FlatpakPolicy policy;

          if (!flatpak_verify_dbus_name (key, error))
            return FALSE;

          policy = flatpak_policy_from_string (value, NULL);
          if ((int) policy != -1)
            flatpak_context_set_session_bus_policy (context, key,
policy);
        }
    }

  if (g_key_file_has_group (metakey,
FLATPAK_METADATA_GROUP_SYSTEM_BUS_POLICY))
    {
      g_auto(GStrv) keys = NULL;
      gsize keys_count;

      keys = g_key_file_get_keys (metakey,
FLATPAK_METADATA_GROUP_SYSTEM_BUS_POLICY, &keys_count, NULL);
      for (i = 0; i < keys_count; i++)
        {
          const char *key = keys[i];
          g_autofree char *value = g_key_file_get_string (metakey,
FLATPAK_METADATA_GROUP_SYSTEM_BUS_POLICY, key, NULL);
          FlatpakPolicy policy;

          if (!flatpak_verify_dbus_name (key, error))
            return FALSE;

          policy = flatpak_policy_from_string (value, NULL);
          if ((int) policy != -1)
            flatpak_context_set_system_bus_policy (context, key, policy);
        }
    }

  if (g_key_file_has_group (metakey, FLATPAK_METADATA_GROUP_ENVIRONMENT))
    {
      g_auto(GStrv) keys = NULL;
      gsize keys_count;

      keys = g_key_file_get_keys (metakey,
FLATPAK_METADATA_GROUP_ENVIRONMENT, &keys_count, NULL);
      for (i = 0; i < keys_count; i++)
        {
          const char *key = keys[i];
          g_autofree char *value = g_key_file_get_string (metakey,
FLATPAK_METADATA_GROUP_ENVIRONMENT, key, NULL);

          flatpak_context_set_env_var (context, key, value);
        }
```

```
    }

  groups = g_key_file_get_groups (metakey, NULL);
  for (i = 0; groups[i] != NULL; i++)
    {
      const char *group = groups[i];
      const char *subsystem;
      int j;

      if (g_str_has_prefix (group, FLATPAK_METADATA_GROUP_PREFIX_POLICY))
        {
          g_auto(GStrv) keys = NULL;
          subsystem = group + strlen
(FLATPAK_METADATA_GROUP_PREFIX_POLICY);
          keys = g_key_file_get_keys (metakey, group, NULL, NULL);
          for (j = 0; keys != NULL && keys[j] != NULL; j++)
            {
              const char *key = keys[j];
              g_autofree char *policy_key = g_strdup_printf ("%s.%s",
subsystem, key);
              g_auto(GStrv) values = NULL;
              int k;

              values = g_key_file_get_string_list (metakey, group, key,
NULL, NULL);
              for (k = 0; values != NULL && values[k] != NULL; k++)
                flatpak_context_apply_generic_policy (context,
policy_key,
                                                      values[k]);
            }
        }
    }

  return TRUE;
}
<sep>
static int mptsas_process_scsi_io_request(MPTSASState *s,
                                          MPIMsgSCSIIORequest *scsi_io,
                                          hwaddr addr)
{
    MPTSASRequest *req;
    MPIMsgSCSIIOReply reply;
    SCSIDevice *sdev;
    int status;

    mptsas_fix_scsi_io_endianness(scsi_io);

    trace_mptsas_process_scsi_io_request(s, scsi_io->Bus, scsi_io-
>TargetID,
                                         scsi_io->LUN[1], scsi_io-
>DataLength);

    status = mptsas_scsi_device_find(s, scsi_io->Bus, scsi_io->TargetID,
                                     scsi_io->LUN, &sdev);
```

```
    if (status) {
        goto bad;
    }

    req = g_new0(MPTSASRequest, 1);
    QTAILQ_INSERT_TAIL(&s->pending, req, next);
    req->scsi_io = *scsi_io;
    req->dev = s;

    status = mptsas_build_sgl(s, req, addr);
    if (status) {
        goto free_bad;
    }

    if (req->qsg.size < scsi_io->DataLength) {
        trace_mptsas_sgl_overflow(s, scsi_io->MsgContext, scsi_io-
>DataLength,
                                  req->qsg.size);
        status = MPI_IOCSTATUS_INVALID_SGL;
        goto free_bad;
    }

    req->sreq = scsi_req_new(sdev, scsi_io->MsgContext,
                             scsi_io->LUN[1], scsi_io->CDB, req);

    if (req->sreq->cmd.xfer > scsi_io->DataLength) {
        goto overrun;
    }
    switch (scsi_io->Control & MPI_SCSIIO_CONTROL_DATADIRECTION_MASK) {
    case MPI_SCSIIO_CONTROL_NODATATRANSFER:
        if (req->sreq->cmd.mode != SCSI_XFER_NONE) {
            goto overrun;
        }
        break;

    case MPI_SCSIIO_CONTROL_WRITE:
        if (req->sreq->cmd.mode != SCSI_XFER_TO_DEV) {
            goto overrun;
        }
        break;

    case MPI_SCSIIO_CONTROL_READ:
        if (req->sreq->cmd.mode != SCSI_XFER_FROM_DEV) {
            goto overrun;
        }
        break;
    }

    if (scsi_req_enqueue(req->sreq)) {
        scsi_req_continue(req->sreq);
    }
    return 0;

overrun:
```

```
    trace_mptsas_scsi_overflow(s, scsi_io->MsgContext, req->sreq-
>cmd.xfer,
                               scsi_io->DataLength);
    status = MPI_IOCSTATUS_SCSI_DATA_OVERRUN;
free_bad:
    mptsas_free_request(req);
bad:
    memset(&reply, 0, sizeof(reply));
    reply.TargetID          = scsi_io->TargetID;
    reply.Bus               = scsi_io->Bus;
    reply.MsgLength         = sizeof(reply) / 4;
    reply.Function          = scsi_io->Function;
    reply.CDBLength         = scsi_io->CDBLength;
    reply.SenseBufferLength = scsi_io->SenseBufferLength;
    reply.MsgContext        = scsi_io->MsgContext;
    reply.SCSIState         = MPI_SCSI_STATE_NO_SCSI_STATUS;
    reply.IOCStatus         = status;

    mptsas_fix_scsi_io_reply_endianness(&reply);
    mptsas_reply(s, (MPIDefaultReply *)&reply);

    return 0;
}
<sep>
static void php_do_pcre_match(INTERNAL_FUNCTION_PARAMETERS, int global)
/* {{{ */
{
      /* parameters */
      char              *regex;               /* Regular expression */
      char              *subject;             /* String to match against
*/
      int                     regex_len;
      int                     subject_len;
      pcre_cache_entry *pce;                  /* Compiled regular
expression */
      zval              *subpats = NULL;/* Array for subpatterns */
      long               flags = 0;           /* Match control flags */
      long               start_offset = 0;    /* Where the new search
starts */

      if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "ss|zll",
&regex, &regex_len,
                                          &subject, &subject_len,
&subpats, &flags, &start_offset) == FAILURE) {
            RETURN_FALSE;
      }

      /* Compile regex or get it from cache. */
      if ((pce = pcre_get_compiled_regex_cache(regex, regex_len
TSRMLS_CC)) == NULL) {
            RETURN_FALSE;
      }
```

```
      php_pcre_match_impl(pce, subject, subject_len, return_value,
subpats,
            global, ZEND_NUM_ARGS() >= 4, flags, start_offset TSRMLS_CC);
}
<sep>
int arch_dup_task_struct(struct task_struct *dst, struct task_struct
*src)
{
      flush_fp_to_thread(src);
      flush_altivec_to_thread(src);
      flush_vsx_to_thread(src);
      flush_spe_to_thread(src);

      *dst = *src;

      clear_task_ebb(dst);

      return 0;
}
<sep>
sign (gcry_mpi_t r, gcry_mpi_t s, gcry_mpi_t input, DSA_secret_key *skey,
      int flags, int hashalgo)
{
  gpg_err_code_t rc;
  gcry_mpi_t hash;
  gcry_mpi_t k;
  gcry_mpi_t kinv;
  gcry_mpi_t tmp;
  const void *abuf;
  unsigned int abits, qbits;
  int extraloops = 0;

  qbits = mpi_get_nbits (skey->q);

  /* Convert the INPUT into an MPI.  */
  rc = _gcry_dsa_normalize_hash (input, &hash, qbits);
  if (rc)
    return rc;

 again:
  /* Create the K value.  */
  if ((flags & PUBKEY_FLAG_RFC6979) && hashalgo)
    {
      /* Use Pornin's method for deterministic DSA.  If this flag is
         set, it is expected that HASH is an opaque MPI with the to be
         signed hash.  That hash is also used as h1 from 3.2.a.  */
      if (!mpi_is_opaque (input))
        {
          rc = GPG_ERR_CONFLICT;
          goto leave;
        }

      abuf = mpi_get_opaque (input, &abits);
      rc = _gcry_dsa_gen_rfc6979_k (&k, skey->q, skey->x,
```

```
                                        abuf, (abits+7)/8, hashalgo,
extraloops);
      if (rc)
        goto leave;
    }
  else
    {
      /* Select a random k with 0 < k < q */
      k = _gcry_dsa_gen_k (skey->q, GCRY_STRONG_RANDOM);
    }

  /* r = (a^k mod p) mod q */
  mpi_powm( r, skey->g, k, skey->p );
  mpi_fdiv_r( r, r, skey->q );

  /* kinv = k^(-1) mod q */
  kinv = mpi_alloc( mpi_get_nlimbs(k) );
  mpi_invm(kinv, k, skey->q );

  /* s = (kinv * ( hash + x * r)) mod q */
  tmp = mpi_alloc( mpi_get_nlimbs(skey->p) );
  mpi_mul( tmp, skey->x, r );
  mpi_add( tmp, tmp, hash );
  mpi_mulm( s , kinv, tmp, skey->q );

  mpi_free(k);
  mpi_free(kinv);
  mpi_free(tmp);

  if (!mpi_cmp_ui (r, 0))
    {
      /* This is a highly unlikely code path.  */
      extraloops++;
      goto again;
    }

  rc = 0;

 leave:
  if (hash != input)
    mpi_free (hash);

  return rc;
}
<sep>
static bool is_topic_in_criterias(
        const char* topic_name,
        const std::vector<Criteria>& criterias)
{
    bool returned_value = false;

    for (auto criteria_it = criterias.begin(); !returned_value &&
            criteria_it != criterias.end(); ++criteria_it)
    {
```

```
        for (auto topic : (*criteria_it).topics)
        {
            if (StringMatching::matchString(topic.c_str(), topic_name))
            {
                returned_value = true;
                break;
            }
        }
    }

    return returned_value;
}
<sep>
char *ad_get_entry(const struct adouble *ad, int eid)
{
      off_t off = ad_getentryoff(ad, eid);
      size_t len = ad_getentrylen(ad, eid);

      if (off == 0 || len == 0) {
            return NULL;
      }

      return ad->ad_data + off;
}
<sep>
static MagickBooleanType WriteTIFFImage(const ImageInfo *image_info,
  Image *image,ExceptionInfo *exception)
{
  const char
    *mode,
    *option;

  CompressionType
    compression;

  EndianType
    endian_type;

  MagickBooleanType
    adjoin,
    debug,
    status;

  MagickOffsetType
    scene;

  QuantumInfo
    *quantum_info;

  QuantumType
    quantum_type;

  register ssize_t
    i;
```

```
  size_t
    imageListLength,
    length;

  ssize_t
    y;

  TIFF
    *tiff;

  TIFFInfo
    tiff_info;

  uint16
    bits_per_sample,
    compress_tag,
    endian,
    photometric,
    predictor;

  unsigned char
    *pixels;

  /*
    Open TIFF file.
  */
  assert(image_info != (const ImageInfo *) NULL);
  assert(image_info->signature == MagickCoreSignature);
  assert(image != (Image *) NULL);
  assert(image->signature == MagickCoreSignature);
  if (image->debug != MagickFalse)
    (void) LogMagickEvent(TraceEvent,GetMagickModule(),"%s",image-
>filename);
  assert(exception != (ExceptionInfo *) NULL);
  assert(exception->signature == MagickCoreSignature);
  status=OpenBlob(image_info,image,WriteBinaryBlobMode,exception);
  if (status == MagickFalse)
    return(status);
  (void) SetMagickThreadValue(tiff_exception,exception);
  endian_type=(HOST_FILLORDER == FILLORDER_LSB2MSB) ? LSBEndian :
MSBEndian;
  option=GetImageOption(image_info,"tiff:endian");
  if (option != (const char *) NULL)
    {
      if (LocaleNCompare(option,"msb",3) == 0)
        endian_type=MSBEndian;
      if (LocaleNCompare(option,"lsb",3) == 0)
        endian_type=LSBEndian;
    }
  mode=endian_type == LSBEndian ? "wl" : "wb";
#if defined(TIFF_VERSION_BIG)
  if (LocaleCompare(image_info->magick,"TIFF64") == 0)
    mode=endian_type == LSBEndian ? "wl8" : "wb8";
```

```
#endif
  tiff=TIFFClientOpen(image->filename,mode,(thandle_t)
image,TIFFReadBlob,
    TIFFWriteBlob,TIFFSeekBlob,TIFFCloseBlob,TIFFGetBlobSize,TIFFMapBlob,
    TIFFUnmapBlob);
  if (tiff == (TIFF *) NULL)
    return(MagickFalse);
  if (exception->severity > ErrorException)
    {
      TIFFClose(tiff);
      return(MagickFalse);
    }
  (void) DeleteImageProfile(image,"tiff:37724");
  scene=0;
  debug=IsEventLogging();
  (void) debug;
  adjoin=image_info->adjoin;
  imageListLength=GetImageListLength(image);
  do
  {
    /*
      Initialize TIFF fields.
    */
    if ((image_info->type != UndefinedType) &&
        (image_info->type != OptimizeType))
      (void) SetImageType(image,image_info->type,exception);
    compression=UndefinedCompression;
    if (image->compression != JPEGCompression)
      compression=image->compression;
    if (image_info->compression != UndefinedCompression)
      compression=image_info->compression;
    switch (compression)
    {
      case FaxCompression:
      case Group4Compression:
      {
        (void) SetImageType(image,BilevelType,exception);
        (void) SetImageDepth(image,1,exception);
        break;
      }
      case JPEGCompression:
      {
        (void) SetImageStorageClass(image,DirectClass,exception);
        (void) SetImageDepth(image,8,exception);
        break;
      }
      default:
        break;
    }
    quantum_info=AcquireQuantumInfo(image_info,image);
    if (quantum_info == (QuantumInfo *) NULL)
      ThrowWriterException(ResourceLimitError,"MemoryAllocationFailed");
    if ((image->storage_class != PseudoClass) && (image->depth >= 32) &&
        (quantum_info->format == UndefinedQuantumFormat) &&
```

```
        (IsHighDynamicRangeImage(image,exception) != MagickFalse))
      {

status=SetQuantumFormat(image,quantum_info,FloatingPointQuantumFormat);
        if (status == MagickFalse)
          {
            quantum_info=DestroyQuantumInfo(quantum_info);

ThrowWriterException(ResourceLimitError,"MemoryAllocationFailed");
          }
      }
    if ((LocaleCompare(image_info->magick,"PTIF") == 0) &&
        (GetPreviousImageInList(image) != (Image *) NULL))
      (void)
TIFFSetField(tiff,TIFFTAG_SUBFILETYPE,FILETYPE_REDUCEDIMAGE);
    if ((image->columns != (uint32) image->columns) ||
        (image->rows != (uint32) image->rows))
      ThrowWriterException(ImageError,"WidthOrHeightExceedsLimit");
    (void) TIFFSetField(tiff,TIFFTAG_IMAGELENGTH,(uint32) image->rows);
    (void) TIFFSetField(tiff,TIFFTAG_IMAGEWIDTH,(uint32) image->columns);
    switch (compression)
    {
      case FaxCompression:
      {
        compress_tag=COMPRESSION_CCITTFAX3;
        option=GetImageOption(image_info,"quantum:polarity");
        if (option == (const char *) NULL)
          SetQuantumMinIsWhite(quantum_info,MagickTrue);
        break;
      }
      case Group4Compression:
      {
        compress_tag=COMPRESSION_CCITTFAX4;
        option=GetImageOption(image_info,"quantum:polarity");
        if (option == (const char *) NULL)
          SetQuantumMinIsWhite(quantum_info,MagickTrue);
        break;
      }
#if defined(COMPRESSION_JBIG)
      case JBIG1Compression:
      {
        compress_tag=COMPRESSION_JBIG;
        break;
      }
#endif
      case JPEGCompression:
      {
        compress_tag=COMPRESSION_JPEG;
        break;
      }
#if defined(COMPRESSION_LZMA)
      case LZMACompression:
      {
        compress_tag=COMPRESSION_LZMA;
```

```
        break;
      }
#endif
      case LZWCompression:
      {
        compress_tag=COMPRESSION_LZW;
        break;
      }
      case RLECompression:
      {
        compress_tag=COMPRESSION_PACKBITS;
        break;
      }
      case ZipCompression:
      {
        compress_tag=COMPRESSION_ADOBE_DEFLATE;
        break;
      }
#if defined(COMPRESSION_ZSTD)
      case ZstdCompression:
      {
        compress_tag=COMPRESSION_ZSTD;
        break;
      }
#endif
      case NoCompression:
      default:
      {
        compress_tag=COMPRESSION_NONE;
        break;
      }
    }
#if defined(MAGICKCORE_HAVE_TIFFISCODECCONFIGURED) || (TIFFLIB_VERSION >
20040919)
    if ((compress_tag != COMPRESSION_NONE) &&
        (TIFFIsCODECConfigured(compress_tag) == 0))
      {
        (void)
ThrowMagickException(exception,GetMagickModule(),CoderError,
          "CompressionNotSupported","`%s'",CommandOptionToMnemonic(
          MagickCompressOptions,(ssize_t) compression));
        compress_tag=COMPRESSION_NONE;
        compression=NoCompression;
      }
#else
      switch (compress_tag)
      {
#if defined(CCITT_SUPPORT)
        case COMPRESSION_CCITTFAX3:
        case COMPRESSION_CCITTFAX4:
#endif
#if defined(YCBCR_SUPPORT) && defined(JPEG_SUPPORT)
        case COMPRESSION_JPEG:
#endif
```

```
#if defined(LZMA_SUPPORT) && defined(COMPRESSION_LZMA)
        case COMPRESSION_LZMA:
#endif
#if defined(LZW_SUPPORT)
        case COMPRESSION_LZW:
#endif
#if defined(PACKBITS_SUPPORT)
        case COMPRESSION_PACKBITS:
#endif
#if defined(ZIP_SUPPORT)
        case COMPRESSION_ADOBE_DEFLATE:
#endif
        case COMPRESSION_NONE:
          break;
        default:
        {
          (void)
ThrowMagickException(exception,GetMagickModule(),CoderError,
            "CompressionNotSupported","`%s'",CommandOptionToMnemonic(
              MagickCompressOptions,(ssize_t) compression));
          compress_tag=COMPRESSION_NONE;
          compression=NoCompression;
          break;
        }
      }
#endif
    if (image->colorspace == CMYKColorspace)
      {
        photometric=PHOTOMETRIC_SEPARATED;
        (void) TIFFSetField(tiff,TIFFTAG_SAMPLESPERPIXEL,4);
        (void) TIFFSetField(tiff,TIFFTAG_INKSET,INKSET_CMYK);
      }
    else
      {
        /*
          Full color TIFF raster.
        */
        if (image->colorspace == LabColorspace)
          {
            photometric=PHOTOMETRIC_CIELAB;
            EncodeLabImage(image,exception);
          }
        else
          if (image->colorspace == YCbCrColorspace)
            {
              photometric=PHOTOMETRIC_YCBCR;
              (void) TIFFSetField(tiff,TIFFTAG_YCBCRSUBSAMPLING,1,1);
              (void) SetImageStorageClass(image,DirectClass,exception);
              (void) SetImageDepth(image,8,exception);
            }
          else
            photometric=PHOTOMETRIC_RGB;
        (void) TIFFSetField(tiff,TIFFTAG_SAMPLESPERPIXEL,3);
        if ((image_info->type != TrueColorType) &&
```

```
        (image_info->type != TrueColorAlphaType))
      {
        if ((image_info->type != PaletteType) &&
            (SetImageGray(image,exception) != MagickFalse))
          {
            photometric=(uint16) (quantum_info->min_is_white !=
              MagickFalse ? PHOTOMETRIC_MINISWHITE :
              PHOTOMETRIC_MINISBLACK);
            (void) TIFFSetField(tiff,TIFFTAG_SAMPLESPERPIXEL,1);
            if ((image->depth == 1) &&
                (image->alpha_trait == UndefinedPixelTrait))
              SetImageMonochrome(image,exception);
          }
        else
          if (image->storage_class == PseudoClass)
            {
              size_t
                depth;

              /*
                Colormapped TIFF raster.
              */
              (void) TIFFSetField(tiff,TIFFTAG_SAMPLESPERPIXEL,1);
              photometric=PHOTOMETRIC_PALETTE;
              depth=1;
              while ((GetQuantumRange(depth)+1) < image->colors)
                depth<<=1;
              status=SetQuantumDepth(image,quantum_info,depth);
              if (status == MagickFalse)
                ThrowWriterException(ResourceLimitError,
                  "MemoryAllocationFailed");
            }
      }
  }
(void) TIFFGetFieldDefaulted(tiff,TIFFTAG_FILLORDER,&endian);
if ((compress_tag == COMPRESSION_CCITTFAX3) ||
    (compress_tag == COMPRESSION_CCITTFAX4))
  {
     if ((photometric != PHOTOMETRIC_MINISWHITE) &&
         (photometric != PHOTOMETRIC_MINISBLACK))
       {
         compress_tag=COMPRESSION_NONE;
         endian=FILLORDER_MSB2LSB;
       }
  }
option=GetImageOption(image_info,"tiff:fill-order");
if (option != (const char *) NULL)
  {
    if (LocaleNCompare(option,"msb",3) == 0)
      endian=FILLORDER_MSB2LSB;
    if (LocaleNCompare(option,"lsb",3) == 0)
      endian=FILLORDER_LSB2MSB;
  }
(void) TIFFSetField(tiff,TIFFTAG_COMPRESSION,compress_tag);
```

```
    (void) TIFFSetField(tiff,TIFFTAG_FILLORDER,endian);
    (void) TIFFSetField(tiff,TIFFTAG_BITSPERSAMPLE,quantum_info->depth);
    if (image->alpha_trait != UndefinedPixelTrait)
      {
        uint16
          extra_samples,
          sample_info[1],
          samples_per_pixel;

        /*
          TIFF has a matte channel.
        */
        extra_samples=1;
        sample_info[0]=EXTRASAMPLE_UNASSALPHA;
        option=GetImageOption(image_info,"tiff:alpha");
        if (option != (const char *) NULL)
          {
            if (LocaleCompare(option,"associated") == 0)
              sample_info[0]=EXTRASAMPLE_ASSOCALPHA;
            else
              if (LocaleCompare(option,"unspecified") == 0)
                sample_info[0]=EXTRASAMPLE_UNSPECIFIED;
          }
        (void) TIFFGetFieldDefaulted(tiff,TIFFTAG_SAMPLESPERPIXEL,
          &samples_per_pixel);
        (void)
TIFFSetField(tiff,TIFFTAG_SAMPLESPERPIXEL,samples_per_pixel+1);
        (void) TIFFSetField(tiff,TIFFTAG_EXTRASAMPLES,extra_samples,
          &sample_info);
        if (sample_info[0] == EXTRASAMPLE_ASSOCALPHA)
          SetQuantumAlphaType(quantum_info,AssociatedQuantumAlpha);
      }
    (void) TIFFSetField(tiff,TIFFTAG_PHOTOMETRIC,photometric);
    switch (quantum_info->format)
    {
      case FloatingPointQuantumFormat:
      {
        (void)
TIFFSetField(tiff,TIFFTAG_SAMPLEFORMAT,SAMPLEFORMAT_IEEEFP);
        (void) TIFFSetField(tiff,TIFFTAG_SMINSAMPLEVALUE,quantum_info-
>minimum);
        (void) TIFFSetField(tiff,TIFFTAG_SMAXSAMPLEVALUE,quantum_info-
>maximum);
        break;
      }
      case SignedQuantumFormat:
      {
        (void) TIFFSetField(tiff,TIFFTAG_SAMPLEFORMAT,SAMPLEFORMAT_INT);
        break;
      }
      case UnsignedQuantumFormat:
      {
        (void) TIFFSetField(tiff,TIFFTAG_SAMPLEFORMAT,SAMPLEFORMAT_UINT);
        break;
```

```
      }
      default:
        break;
    }
    (void) TIFFSetField(tiff,TIFFTAG_PLANARCONFIG,PLANARCONFIG_CONTIG);
    if (photometric == PHOTOMETRIC_RGB)
      if ((image_info->interlace == PlaneInterlace) ||
          (image_info->interlace == PartitionInterlace))
        (void)
TIFFSetField(tiff,TIFFTAG_PLANARCONFIG,PLANARCONFIG_SEPARATE);
    predictor=0;
    switch (compress_tag)
    {
      case COMPRESSION_JPEG:
      {
#if defined(JPEG_SUPPORT)
        if (image_info->quality != UndefinedCompressionQuality)
          (void) TIFFSetField(tiff,TIFFTAG_JPEGQUALITY,image_info-
>quality);
        (void)
TIFFSetField(tiff,TIFFTAG_JPEGCOLORMODE,JPEGCOLORMODE_RAW);
        if (IssRGBCompatibleColorspace(image->colorspace) != MagickFalse)
          {
            const char
              *value;

            (void)
TIFFSetField(tiff,TIFFTAG_JPEGCOLORMODE,JPEGCOLORMODE_RGB);
            if (image->colorspace == YCbCrColorspace)
              {
                const char
                  *sampling_factor;

                GeometryInfo
                  geometry_info;

                MagickStatusType
                  flags;

                sampling_factor=(const char *) NULL;
                value=GetImageProperty(image,"jpeg:sampling-
factor",exception);
                if (value != (char *) NULL)
                  {
                    sampling_factor=value;
                    if (image->debug != MagickFalse)
                      (void) LogMagickEvent(CoderEvent,GetMagickModule(),
                        "  Input sampling-factors=%s",sampling_factor);
                  }
                if (image_info->sampling_factor != (char *) NULL)
                  sampling_factor=image_info->sampling_factor;
                if (sampling_factor != (const char *) NULL)
                  {
                    flags=ParseGeometry(sampling_factor,&geometry_info);
```

```
                    if ((flags & SigmaValue) == 0)
                      geometry_info.sigma=geometry_info.rho;
                    (void)
TIFFSetField(tiff,TIFFTAG_YCBCRSUBSAMPLING,(uint16)
                      geometry_info.rho,(uint16) geometry_info.sigma);
                  }
                }
          }
        (void) TIFFGetFieldDefaulted(tiff,TIFFTAG_BITSPERSAMPLE,
          &bits_per_sample);
        if (bits_per_sample == 12)
          (void)
TIFFSetField(tiff,TIFFTAG_JPEGTABLESMODE,JPEGTABLESMODE_QUANT);
#endif
        break;
      }
      case COMPRESSION_ADOBE_DEFLATE:
      {
        (void) TIFFGetFieldDefaulted(tiff,TIFFTAG_BITSPERSAMPLE,
          &bits_per_sample);
        if (((photometric == PHOTOMETRIC_RGB) ||
             (photometric == PHOTOMETRIC_SEPARATED) ||
             (photometric == PHOTOMETRIC_MINISBLACK)) &&
            ((bits_per_sample == 8) || (bits_per_sample == 16)))
          predictor=PREDICTOR_HORIZONTAL;
        (void) TIFFSetField(tiff,TIFFTAG_ZIPQUALITY,(long) (
          image_info->quality == UndefinedCompressionQuality ? 7 :
          MagickMin((ssize_t) image_info->quality/10,9)));
        break;
      }
      case COMPRESSION_CCITTFAX3:
      {
        /*
          Byte-aligned EOL.
        */
        (void) TIFFSetField(tiff,TIFFTAG_GROUP3OPTIONS,4);
        break;
      }
      case COMPRESSION_CCITTFAX4:
        break;
#if defined(LZMA_SUPPORT) && defined(COMPRESSION_LZMA)
      case COMPRESSION_LZMA:
      {
        if (((photometric == PHOTOMETRIC_RGB) ||
             (photometric == PHOTOMETRIC_SEPARATED) ||
             (photometric == PHOTOMETRIC_MINISBLACK)) &&
            ((bits_per_sample == 8) || (bits_per_sample == 16)))
          predictor=PREDICTOR_HORIZONTAL;
        (void) TIFFSetField(tiff,TIFFTAG_LZMAPRESET,(long) (
          image_info->quality == UndefinedCompressionQuality ? 7 :
          MagickMin((ssize_t) image_info->quality/10,9)));
        break;
      }
#endif
```

```
      case COMPRESSION_LZW:
      {
        (void) TIFFGetFieldDefaulted(tiff,TIFFTAG_BITSPERSAMPLE,
          &bits_per_sample);
        if (((photometric == PHOTOMETRIC_RGB) ||
             (photometric == PHOTOMETRIC_SEPARATED) ||
             (photometric == PHOTOMETRIC_MINISBLACK)) &&
            ((bits_per_sample == 8) || (bits_per_sample == 16)))
          predictor=PREDICTOR_HORIZONTAL;
        break;
      }
#if defined(WEBP_SUPPORT) && defined(COMPRESSION_WEBP)
      case COMPRESSION_WEBP:
      {
        (void) TIFFGetFieldDefaulted(tiff,TIFFTAG_BITSPERSAMPLE,
          &bits_per_sample);
        if (((photometric == PHOTOMETRIC_RGB) ||
             (photometric == PHOTOMETRIC_SEPARATED) ||
             (photometric == PHOTOMETRIC_MINISBLACK)) &&
            ((bits_per_sample == 8) || (bits_per_sample == 16)))
          predictor=PREDICTOR_HORIZONTAL;
        (void) TIFFSetField(tiff,TIFFTAG_WEBP_LEVEL,image_info->quality);
        if (image_info->quality >= 100)
          (void) TIFFSetField(tiff,TIFFTAG_WEBP_LOSSLESS,1);
        break;
      }
#endif
#if defined(ZSTD_SUPPORT) && defined(COMPRESSION_ZSTD)
      case COMPRESSION_ZSTD:
      {
        (void) TIFFGetFieldDefaulted(tiff,TIFFTAG_BITSPERSAMPLE,
          &bits_per_sample);
        if (((photometric == PHOTOMETRIC_RGB) ||
             (photometric == PHOTOMETRIC_SEPARATED) ||
             (photometric == PHOTOMETRIC_MINISBLACK)) &&
            ((bits_per_sample == 8) || (bits_per_sample == 16)))
          predictor=PREDICTOR_HORIZONTAL;
        (void) TIFFSetField(tiff,TIFFTAG_ZSTD_LEVEL,22*image_info-
>quality/
          100.0);
        break;
      }
#endif
      default:
        break;
    }
    option=GetImageOption(image_info,"tiff:predictor");
    if (option != (const char * ) NULL)
      predictor=(uint16) strtol(option,(char **) NULL,10);
    if (predictor != 0)
      (void) TIFFSetField(tiff,TIFFTAG_PREDICTOR,predictor);
    if ((image->resolution.x != 0.0) && (image->resolution.y != 0.0))
      {
        unsigned short
```

```
          units;

        /*
          Set image resolution.
        */
        units=RESUNIT_NONE;
        if (image->units == PixelsPerInchResolution)
          units=RESUNIT_INCH;
        if (image->units == PixelsPerCentimeterResolution)
          units=RESUNIT_CENTIMETER;
        (void) TIFFSetField(tiff,TIFFTAG_RESOLUTIONUNIT,(uint16) units);
        (void) TIFFSetField(tiff,TIFFTAG_XRESOLUTION,image-
>resolution.x);
        (void) TIFFSetField(tiff,TIFFTAG_YRESOLUTION,image-
>resolution.y);
        if ((image->page.x < 0) || (image->page.y < 0))
          (void)
ThrowMagickException(exception,GetMagickModule(),CoderError,
            "TIFF: negative image positions unsupported","%s",image-
>filename);
        if ((image->page.x > 0) && (image->resolution.x > 0.0))
          {
            /*
              Set horizontal image position.
            */
            (void) TIFFSetField(tiff,TIFFTAG_XPOSITION,(float) image-
>page.x/
              image->resolution.x);
          }
        if ((image->page.y > 0) && (image->resolution.y > 0.0))
          {
            /*
              Set vertical image position.
            */
            (void) TIFFSetField(tiff,TIFFTAG_YPOSITION,(float) image-
>page.y/
              image->resolution.y);
          }
      }
    if (image->chromaticity.white_point.x != 0.0)
      {
        float
          chromaticity[6];

        /*
          Set image chromaticity.
        */
        chromaticity[0]=(float) image->chromaticity.red_primary.x;
        chromaticity[1]=(float) image->chromaticity.red_primary.y;
        chromaticity[2]=(float) image->chromaticity.green_primary.x;
        chromaticity[3]=(float) image->chromaticity.green_primary.y;
        chromaticity[4]=(float) image->chromaticity.blue_primary.x;
        chromaticity[5]=(float) image->chromaticity.blue_primary.y;
```

```
        (void)
TIFFSetField(tiff,TIFFTAG_PRIMARYCHROMATICITIES,chromaticity);
        chromaticity[0]=(float) image->chromaticity.white_point.x;
        chromaticity[1]=(float) image->chromaticity.white_point.y;
        (void) TIFFSetField(tiff,TIFFTAG_WHITEPOINT,chromaticity);
      }
    option=GetImageOption(image_info,"tiff:write-layers");
    if (IsStringTrue(option) != MagickFalse)
      {
        (void)
TIFFWritePhotoshopLayers(image,image_info,endian_type,exception);
        adjoin=MagickFalse;
      }
    if ((LocaleCompare(image_info->magick,"PTIF") != 0) &&
        (adjoin != MagickFalse) && (imageListLength > 1))
      {
        (void) TIFFSetField(tiff,TIFFTAG_SUBFILETYPE,FILETYPE_PAGE);
        if (image->scene != 0)
          (void) TIFFSetField(tiff,TIFFTAG_PAGENUMBER,(uint16) image-
>scene,
            imageListLength);
      }
    if (image->orientation != UndefinedOrientation)
      (void) TIFFSetField(tiff,TIFFTAG_ORIENTATION,(uint16) image-
>orientation);
    else
      (void) TIFFSetField(tiff,TIFFTAG_ORIENTATION,ORIENTATION_TOPLEFT);
    TIFFSetProfiles(tiff,image);
    {
      uint16
        page,
        pages;

      page=(uint16) scene;
      pages=(uint16) imageListLength;
      if ((LocaleCompare(image_info->magick,"PTIF") != 0) &&
          (adjoin != MagickFalse) && (pages > 1))
        (void) TIFFSetField(tiff,TIFFTAG_SUBFILETYPE,FILETYPE_PAGE);
      (void) TIFFSetField(tiff,TIFFTAG_PAGENUMBER,page,pages);
    }
    (void) TIFFSetProperties(tiff,adjoin,image,exception);
DisableMSCWarning(4127)
    if (0)
RestoreMSCWarning
      (void) TIFFSetEXIFProperties(tiff,image,exception);
    /*
      Write image scanlines.
    */
    if (GetTIFFInfo(image_info,tiff,&tiff_info) == MagickFalse)
      ThrowWriterException(ResourceLimitError,"MemoryAllocationFailed");
    quantum_info->endian=LSBEndian;
    pixels=(unsigned char *) GetQuantumPixels(quantum_info);
    tiff_info.scanline=(unsigned char *) GetQuantumPixels(quantum_info);
    switch (photometric)
```

```
    {
      case PHOTOMETRIC_CIELAB:
      case PHOTOMETRIC_YCBCR:
      case PHOTOMETRIC_RGB:
      {
        /*
          RGB TIFF image.
        */
        switch (image_info->interlace)
        {
          case NoInterlace:
          default:
          {
            quantum_type=RGBQuantum;
            if (image->alpha_trait != UndefinedPixelTrait)
              quantum_type=RGBAQuantum;
            for (y=0; y < (ssize_t) image->rows; y++)
            {
              register const Quantum
                *magick_restrict p;

              p=GetVirtualPixels(image,0,y,image->columns,1,exception);
              if (p == (const Quantum *) NULL)
                break;
              length=ExportQuantumPixels(image,(CacheView *)
NULL,quantum_info,
                quantum_type,pixels,exception);
              (void) length;
              if (TIFFWritePixels(tiff,&tiff_info,y,0,image) == -1)
                break;
              if (image->previous == (Image *) NULL)
                {

status=SetImageProgress(image,SaveImageTag,(MagickOffsetType)
                    y,image->rows);
                  if (status == MagickFalse)
                    break;
                }
            }
            break;
          }
          case PlaneInterlace:
          case PartitionInterlace:
          {
            /*
              Plane interlacing:  RRRRRR...GGGGGG...BBBBBB...
            */
            for (y=0; y < (ssize_t) image->rows; y++)
            {
              register const Quantum
                *magick_restrict p;

              p=GetVirtualPixels(image,0,y,image->columns,1,exception);
              if (p == (const Quantum *) NULL)
```

```
              break;
            length=ExportQuantumPixels(image,(CacheView *)
NULL,quantum_info,
              RedQuantum,pixels,exception);
            if (TIFFWritePixels(tiff,&tiff_info,y,0,image) == -1)
              break;
          }
          if (image->previous == (Image *) NULL)
            {
              status=SetImageProgress(image,SaveImageTag,100,400);
              if (status == MagickFalse)
                break;
            }
          for (y=0; y < (ssize_t) image->rows; y++)
          {
            register const Quantum
              *magick_restrict p;

            p=GetVirtualPixels(image,0,y,image->columns,1,exception);
            if (p == (const Quantum *) NULL)
              break;
            length=ExportQuantumPixels(image,(CacheView *)
NULL,quantum_info,
              GreenQuantum,pixels,exception);
            if (TIFFWritePixels(tiff,&tiff_info,y,1,image) == -1)
              break;
          }
          if (image->previous == (Image *) NULL)
            {
              status=SetImageProgress(image,SaveImageTag,200,400);
              if (status == MagickFalse)
                break;
            }
          for (y=0; y < (ssize_t) image->rows; y++)
          {
            register const Quantum
              *magick_restrict p;

            p=GetVirtualPixels(image,0,y,image->columns,1,exception);
            if (p == (const Quantum *) NULL)
              break;
            length=ExportQuantumPixels(image,(CacheView *)
NULL,quantum_info,
              BlueQuantum,pixels,exception);
            if (TIFFWritePixels(tiff,&tiff_info,y,2,image) == -1)
              break;
          }
          if (image->previous == (Image *) NULL)
            {
              status=SetImageProgress(image,SaveImageTag,300,400);
              if (status == MagickFalse)
                break;
            }
          if (image->alpha_trait != UndefinedPixelTrait)
```

```
              for (y=0; y < (ssize_t) image->rows; y++)
              {
                register const Quantum
                  *magick_restrict p;

                p=GetVirtualPixels(image,0,y,image->columns,1,exception);
                if (p == (const Quantum *) NULL)
                  break;
                length=ExportQuantumPixels(image,(CacheView *) NULL,
                  quantum_info,AlphaQuantum,pixels,exception);
                if (TIFFWritePixels(tiff,&tiff_info,y,3,image) == -1)
                  break;
              }
            if (image->previous == (Image *) NULL)
              {
                status=SetImageProgress(image,SaveImageTag,400,400);
                if (status == MagickFalse)
                  break;
              }
            break;
          }
        }
        break;
      }
      case PHOTOMETRIC_SEPARATED:
      {
        /*
          CMYK TIFF image.
        */
        quantum_type=CMYKQuantum;
        if (image->alpha_trait != UndefinedPixelTrait)
          quantum_type=CMYKAQuantum;
        if (image->colorspace != CMYKColorspace)
          (void)
TransformImageColorspace(image,CMYKColorspace,exception);
        for (y=0; y < (ssize_t) image->rows; y++)
        {
          register const Quantum
            *magick_restrict p;

          p=GetVirtualPixels(image,0,y,image->columns,1,exception);
          if (p == (const Quantum *) NULL)
            break;
          length=ExportQuantumPixels(image,(CacheView *)
NULL,quantum_info,
            quantum_type,pixels,exception);
          if (TIFFWritePixels(tiff,&tiff_info,y,0,image) == -1)
            break;
          if (image->previous == (Image *) NULL)
            {

status=SetImageProgress(image,SaveImageTag,(MagickOffsetType) y,
                image->rows);
              if (status == MagickFalse)
```

```
              break;
            }
        }
        break;
      }
      case PHOTOMETRIC_PALETTE:
      {
        uint16
          *blue,
          *green,
          *red;

        /*
          Colormapped TIFF image.
        */
        red=(uint16 *) AcquireQuantumMemory(65536,sizeof(*red));
        green=(uint16 *) AcquireQuantumMemory(65536,sizeof(*green));
        blue=(uint16 *) AcquireQuantumMemory(65536,sizeof(*blue));
        if ((red == (uint16 *) NULL) || (green == (uint16 *) NULL) ||
            (blue == (uint16 *) NULL))
          {
            if (red != (uint16 *) NULL)
              red=(uint16 *) RelinquishMagickMemory(red);
            if (green != (uint16 *) NULL)
              green=(uint16 *) RelinquishMagickMemory(green);
            if (blue != (uint16 *) NULL)
              blue=(uint16 *) RelinquishMagickMemory(blue);

ThrowWriterException(ResourceLimitError,"MemoryAllocationFailed");
          }
        /*
          Initialize TIFF colormap.
        */
        (void) memset(red,0,65536*sizeof(*red));
        (void) memset(green,0,65536*sizeof(*green));
        (void) memset(blue,0,65536*sizeof(*blue));
        for (i=0; i < (ssize_t) image->colors; i++)
        {
          red[i]=ScaleQuantumToShort(image->colormap[i].red);
          green[i]=ScaleQuantumToShort(image->colormap[i].green);
          blue[i]=ScaleQuantumToShort(image->colormap[i].blue);
        }
        (void) TIFFSetField(tiff,TIFFTAG_COLORMAP,red,green,blue);
        red=(uint16 *) RelinquishMagickMemory(red);
        green=(uint16 *) RelinquishMagickMemory(green);
        blue=(uint16 *) RelinquishMagickMemory(blue);
      }
      default:
      {
        /*
          Convert PseudoClass packets to contiguous grayscale scanlines.
        */
        quantum_type=IndexQuantum;
        if (image->alpha_trait != UndefinedPixelTrait)
```

```
          {
            if (photometric != PHOTOMETRIC_PALETTE)
              quantum_type=GrayAlphaQuantum;
            else
              quantum_type=IndexAlphaQuantum;
           }
         else
           if (photometric != PHOTOMETRIC_PALETTE)
            quantum_type=GrayQuantum;
        for (y=0; y < (ssize_t) image->rows; y++)
        {
          register const Quantum
            *magick_restrict p;

          p=GetVirtualPixels(image,0,y,image->columns,1,exception);
          if (p == (const Quantum *) NULL)
            break;
          length=ExportQuantumPixels(image,(CacheView *)
NULL,quantum_info,
            quantum_type,pixels,exception);
          if (TIFFWritePixels(tiff,&tiff_info,y,0,image) == -1)
            break;
          if (image->previous == (Image *) NULL)
            {

status=SetImageProgress(image,SaveImageTag,(MagickOffsetType) y,
                image->rows);
              if (status == MagickFalse)
                break;
            }
        }
        break;
      }
    }
    quantum_info=DestroyQuantumInfo(quantum_info);
    if (image->colorspace == LabColorspace)
      DecodeLabImage(image,exception);
    DestroyTIFFInfo(&tiff_info);
DisableMSCWarning(4127)
    if (0 && (image_info->verbose != MagickFalse))
RestoreMSCWarning
      TIFFPrintDirectory(tiff,stdout,MagickFalse);
    (void) TIFFWriteDirectory(tiff);
    image=SyncNextImageInList(image);
    if (image == (Image *) NULL)
      break;
    status=SetImageProgress(image,SaveImagesTag,scene++,imageListLength);
    if (status == MagickFalse)
      break;
  } while (adjoin != MagickFalse);
  TIFFClose(tiff);
  return(MagickTrue);
}
<sep>
```

```
static uint get_table_structure(char *table, char *db, char *table_type,
                                char *ignore_flag)
{
  my_bool    init=0, delayed, write_data, complete_insert;
  my_ulonglong num_fields;
  char       *result_table, *opt_quoted_table;
  const char *insert_option;
  char          name_buff[NAME_LEN+3],table_buff[NAME_LEN*2+3];
  char       table_buff2[NAME_LEN*2+3], query_buff[QUERY_LENGTH];
  const char *show_fields_stmt= "SELECT `COLUMN_NAME` AS `Field`, "
                                "`COLUMN_TYPE` AS `Type`, "
                                "`IS_NULLABLE` AS `Null`, "
                                "`COLUMN_KEY` AS `Key`, "
                                "`COLUMN_DEFAULT` AS `Default`, "
                                "`EXTRA` AS `Extra`, "
                                "`COLUMN_COMMENT` AS `Comment` "
                                "FROM `INFORMATION_SCHEMA`.`COLUMNS`
WHERE "
                                "TABLE_SCHEMA = '%s' AND TABLE_NAME =
'%s'";
  FILE       *sql_file= md_result_file;
  int        len;
  my_bool    is_log_table;
  MYSQL_RES  *result;
  MYSQL_ROW  row;
  DBUG_ENTER("get_table_structure");
  DBUG_PRINT("enter", ("db: %s  table: %s", db, table));

  *ignore_flag= check_if_ignore_table(table, table_type);

  delayed= opt_delayed;
  if (delayed && (*ignore_flag & IGNORE_INSERT_DELAYED))
  {
    delayed= 0;
    verbose_msg("-- Warning: Unable to use delayed inserts for table '%s'
"
                "because it's of type %s\n", table, table_type);
  }

  complete_insert= 0;
  if ((write_data= !(*ignore_flag & IGNORE_DATA)))
  {
    complete_insert= opt_complete_insert;
    if (!insert_pat_inited)
    {
      insert_pat_inited= 1;
      init_dynamic_string_checked(&insert_pat, "", 1024, 1024);
    }
    else
      dynstr_set_checked(&insert_pat, "");
  }

  insert_option= ((delayed && opt_ignore) ? " DELAYED IGNORE " :
                  delayed ? " DELAYED " : opt_ignore ? " IGNORE " : "");
```

```
  verbose_msg("-- Retrieving table structure for table %s...\n", table);

  len= my_snprintf(query_buff, sizeof(query_buff),
                   "SET SQL_QUOTE_SHOW_CREATE=%d",
                   (opt_quoted || opt_keywords));
  if (!create_options)
    strmov(query_buff+len,
           "/*!40102 ,SQL_MODE=concat(@@sql_mode, _utf8
',NO_KEY_OPTIONS,NO_TABLE_OPTIONS,NO_FIELD_OPTIONS') */");

  result_table=     quote_name(table, table_buff, 1);
  opt_quoted_table= quote_name(table, table_buff2, 0);

  if (opt_order_by_primary)
    order_by= primary_key_fields(result_table);

  if (!opt_xml && !mysql_query_with_error_report(mysql, 0, query_buff))
  {
    /* using SHOW CREATE statement */
    if (!opt_no_create_info)
    {
      /* Make an sql-file, if path was given iow. option -T was given */
      char buff[20+FN_REFLEN];
      MYSQL_FIELD *field;

      my_snprintf(buff, sizeof(buff), "show create table %s",
result_table);

      if (switch_character_set_results(mysql, "binary") ||
          mysql_query_with_error_report(mysql, &result, buff) ||
          switch_character_set_results(mysql, default_charset))
        DBUG_RETURN(0);

      if (path)
      {
        if (!(sql_file= open_sql_file_for_table(table, O_WRONLY)))
          DBUG_RETURN(0);

        write_header(sql_file, db);
      }

      if (strcmp (table_type, "VIEW") == 0)         /* view */
        print_comment(sql_file, 0,
                      "\n--\n-- Temporary table structure for view %s\n--
\n\n",
                      fix_identifier_with_newline(result_table));
      else
        print_comment(sql_file, 0,
                      "\n--\n-- Table structure for table %s\n--\n\n",
                      fix_identifier_with_newline(result_table));

      if (opt_drop)
      {
```

```
      /*
        Even if the "table" is a view, we do a DROP TABLE here.  The
        view-specific code below fills in the DROP VIEW.
        We will skip the DROP TABLE for general_log and slow_log, since
        those stmts will fail, in case we apply dump by enabling logging.
       */
        if (!general_log_or_slow_log_tables(db, table))
          fprintf(sql_file, "DROP TABLE IF EXISTS %s;\n",
                  opt_quoted_table);
        check_io(sql_file);
      }

      field= mysql_fetch_field_direct(result, 0);
      if (strcmp(field->name, "View") == 0)
      {
        char *scv_buff= NULL;
        my_ulonglong n_cols;

        verbose_msg("-- It's a view, create dummy table for view\n");

        /* save "show create" statement for later */
        if ((row= mysql_fetch_row(result)) && (scv_buff=row[1]))
          scv_buff= my_strdup(scv_buff, MYF(0));

        mysql_free_result(result);

        /*
          Create a table with the same name as the view and with columns
of
          the same name in order to satisfy views that depend on this
view.
          The table will be removed when the actual view is created.

          The properties of each column, are not preserved in this
temporary
          table, because they are not necessary.

          This will not be necessary once we can determine dependencies
          between views and can simply dump them in the appropriate
order.
        */
        my_snprintf(query_buff, sizeof(query_buff),
                    "SHOW FIELDS FROM %s", result_table);
        if (switch_character_set_results(mysql, "binary") ||
            mysql_query_with_error_report(mysql, &result, query_buff) ||
            switch_character_set_results(mysql, default_charset))
        {
          /*
            View references invalid or privileged table/col/fun (err
1356),
            so we cannot create a stand-in table.  Be defensive and dump
            a comment with the view's 'show create' statement. (Bug
#17371)
          */
```

```
          if (mysql_errno(mysql) == ER_VIEW_INVALID)
            fprintf(sql_file, "\n-- failed on view %s: %s\n\n",
result_table, scv_buff ? scv_buff : "");

          my_free(scv_buff);

          DBUG_RETURN(0);
        }
        else
          my_free(scv_buff);

        n_cols= mysql_num_rows(result);
        if (0 != n_cols)
        {

          /*
            The actual formula is based on the column names and how the
.FRM
            files are stored and is too volatile to be repeated here.
            Thus we simply warn the user if the columns exceed a limit we
            know works most of the time.
          */
          if (n_cols >= 1000)
            fprintf(stderr,
                    "-- Warning: Creating a stand-in table for view %s
may"
                    " fail when replaying the dump file produced because
"
                    "of the number of columns exceeding 1000. Exercise "
                    "caution when replaying the produced dump file.\n",
                    table);
          if (opt_drop)
          {
            /*
              We have already dropped any table of the same name above,
so
              here we just drop the view.
            */

            fprintf(sql_file, "/*!50001 DROP VIEW IF EXISTS %s*/;\n",
                    opt_quoted_table);
            check_io(sql_file);
          }

          fprintf(sql_file,
                  "SET @saved_cs_client     = @@character_set_client;\n"
                  "SET character_set_client = utf8;\n"
                  "/*!50001 CREATE TABLE %s (\n",
                  result_table);

          /*
            Get first row, following loop will prepend comma - keeps from
```

```
            having to know if the row being printed is last to determine
if
            there should be a _trailing_ comma.
          */

          row= mysql_fetch_row(result);

          /*
            The actual column type doesn't matter anyway, since the table
will
            be dropped at run time.
            We do tinyint to avoid hitting the row size limit.
          */
          fprintf(sql_file, "  %s tinyint NOT NULL",
                  quote_name(row[0], name_buff, 0));

          while((row= mysql_fetch_row(result)))
          {
            /* col name, col type */
            fprintf(sql_file, ",\n  %s tinyint NOT NULL",
                    quote_name(row[0], name_buff, 0));
          }

          /*
            Stand-in tables are always MyISAM tables as the default
            engine might have a column-limit that's lower than the
            number of columns in the view, and MyISAM support is
            guaranteed to be in the server anyway.
          */
          fprintf(sql_file,
                  "\n) ENGINE=MyISAM */;\n"
                  "SET character_set_client = @saved_cs_client;\n");

          check_io(sql_file);
        }

        mysql_free_result(result);

        if (path)
          my_fclose(sql_file, MYF(MY_WME));

        seen_views= 1;
        DBUG_RETURN(0);
      }

      row= mysql_fetch_row(result);

      is_log_table= general_log_or_slow_log_tables(db, table);
      if (is_log_table)
        row[1]+= 13; /* strlen("CREATE TABLE ")= 13 */
      if (opt_compatible_mode & 3)
      {
        fprintf(sql_file,
```

```
                is_log_table ? "CREATE TABLE IF NOT EXISTS %s;\n" :
"%s;\n",
                row[1]);
      }
      else
      {
        fprintf(sql_file,
                "/*!40101 SET @saved_cs_client     =
@@character_set_client */;\n"
                "/*!40101 SET character_set_client = utf8 */;\n"
                "%s%s;\n"
                "/*!40101 SET character_set_client = @saved_cs_client
*/;\n",
                is_log_table ? "CREATE TABLE IF NOT EXISTS " : "",
                row[1]);
      }

      check_io(sql_file);
      mysql_free_result(result);
    }
    my_snprintf(query_buff, sizeof(query_buff), "show fields from %s",
                result_table);
    if (mysql_query_with_error_report(mysql, &result, query_buff))
    {
      if (path)
        my_fclose(sql_file, MYF(MY_WME));
      DBUG_RETURN(0);
    }

    /*
      If write_data is true, then we build up insert statements for
      the table's data. Note: in subsequent lines of code, this test
      will have to be performed each time we are appending to
      insert_pat.
    */
    if (write_data)
    {
      if (opt_replace_into)
        dynstr_append_checked(&insert_pat, "REPLACE ");
      else
        dynstr_append_checked(&insert_pat, "INSERT ");
      dynstr_append_checked(&insert_pat, insert_option);
      dynstr_append_checked(&insert_pat, "INTO ");
      dynstr_append_checked(&insert_pat, opt_quoted_table);
      if (complete_insert)
      {
        dynstr_append_checked(&insert_pat, " (");
      }
      else
      {
        dynstr_append_checked(&insert_pat, " VALUES ");
        if (!extended_insert)
          dynstr_append_checked(&insert_pat, "(");
      }
```

```
    }

    while ((row= mysql_fetch_row(result)))
    {
      if (complete_insert)
      {
        if (init)
        {
          dynstr_append_checked(&insert_pat, ", ");
        }
        init=1;
        dynstr_append_checked(&insert_pat,
                      quote_name(row[SHOW_FIELDNAME], name_buff, 0));
      }
    }
    num_fields= mysql_num_rows(result);
    mysql_free_result(result);
  }
  else
  {
    verbose_msg("%s: Warning: Can't set SQL_QUOTE_SHOW_CREATE option
(%s)\n",
                my_progname, mysql_error(mysql));

    my_snprintf(query_buff, sizeof(query_buff), show_fields_stmt, db,
table);

    if (mysql_query_with_error_report(mysql, &result, query_buff))
      DBUG_RETURN(0);

    /* Make an sql-file, if path was given iow. option -T was given */
    if (!opt_no_create_info)
    {
      if (path)
      {
        if (!(sql_file= open_sql_file_for_table(table, O_WRONLY)))
          DBUG_RETURN(0);
        write_header(sql_file, db);
      }

      print_comment(sql_file, 0,
                    "\n--\n-- Table structure for table %s\n--\n\n",
                    fix_identifier_with_newline(result_table));
      if (opt_drop)
        fprintf(sql_file, "DROP TABLE IF EXISTS %s;\n", result_table);
      if (!opt_xml)
        fprintf(sql_file, "CREATE TABLE %s (\n", result_table);
      else
        print_xml_tag(sql_file, "\t", "\n", "table_structure", "name=",
table,
                  NullS);
      check_io(sql_file);
    }
```

```
if (write_data)
{
  if (opt_replace_into)
    dynstr_append_checked(&insert_pat, "REPLACE ");
  else
    dynstr_append_checked(&insert_pat, "INSERT ");
  dynstr_append_checked(&insert_pat, insert_option);
  dynstr_append_checked(&insert_pat, "INTO ");
  dynstr_append_checked(&insert_pat, result_table);
  if (complete_insert)
    dynstr_append_checked(&insert_pat, " (");
  else
  {
    dynstr_append_checked(&insert_pat, " VALUES ");
    if (!extended_insert)
      dynstr_append_checked(&insert_pat, "(");
  }
}

while ((row= mysql_fetch_row(result)))
{
  ulong *lengths= mysql_fetch_lengths(result);
  if (init)
  {
    if (!opt_xml && !opt_no_create_info)
    {
      fputs(",\n",sql_file);
      check_io(sql_file);
    }
    if (complete_insert)
      dynstr_append_checked(&insert_pat, ", ");
  }
  init=1;
  if (complete_insert)
    dynstr_append_checked(&insert_pat,
                  quote_name(row[SHOW_FIELDNAME], name_buff, 0));
  if (!opt_no_create_info)
  {
    if (opt_xml)
    {
      print_xml_row(sql_file, "field", result, &row, NullS);
      continue;
    }

    if (opt_keywords)
      fprintf(sql_file, "  %s.%s %s", result_table,
              quote_name(row[SHOW_FIELDNAME],name_buff, 0),
              row[SHOW_TYPE]);
    else
      fprintf(sql_file, "  %s %s", quote_name(row[SHOW_FIELDNAME],
                                              name_buff, 0),
              row[SHOW_TYPE]);
    if (row[SHOW_DEFAULT])
    {
```

```
          fputs(" DEFAULT ", sql_file);
          unescape(sql_file, row[SHOW_DEFAULT], lengths[SHOW_DEFAULT]);
        }
        if (!row[SHOW_NULL][0])
          fputs(" NOT NULL", sql_file);
        if (row[SHOW_EXTRA][0])
          fprintf(sql_file, " %s",row[SHOW_EXTRA]);
        check_io(sql_file);
      }
    }
    num_fields= mysql_num_rows(result);
    mysql_free_result(result);
    if (!opt_no_create_info)
    {
      /* Make an sql-file, if path was given iow. option -T was given */
      char buff[20+FN_REFLEN];
      uint keynr,primary_key;
      my_snprintf(buff, sizeof(buff), "show keys from %s", result_table);
      if (mysql_query_with_error_report(mysql, &result, buff))
      {
        if (mysql_errno(mysql) == ER_WRONG_OBJECT)
        {
          /* it is VIEW */
          fputs("\t\t<options Comment=\"view\" />\n", sql_file);
          goto continue_xml;
        }
        fprintf(stderr, "%s: Can't get keys for table %s (%s)\n",
                my_progname, result_table, mysql_error(mysql));
        if (path)
          my_fclose(sql_file, MYF(MY_WME));
        DBUG_RETURN(0);
      }

      /* Find first which key is primary key */
      keynr=0;
      primary_key=INT_MAX;
      while ((row= mysql_fetch_row(result)))
      {
        if (atoi(row[3]) == 1)
        {
          keynr++;
#ifdef FORCE_PRIMARY_KEY
          if (atoi(row[1]) == 0 && primary_key == INT_MAX)
            primary_key=keynr;
#endif
          if (!strcmp(row[2],"PRIMARY"))
          {
            primary_key=keynr;
            break;
          }
        }
      }
      mysql_data_seek(result,0);
      keynr=0;
```

```
      while ((row= mysql_fetch_row(result)))
      {
        if (opt_xml)
        {
          print_xml_row(sql_file, "key", result, &row, NullS);
          continue;
        }

        if (atoi(row[3]) == 1)
        {
          if (keynr++)
            putc(')', sql_file);
          if (atoi(row[1]))       /* Test if duplicate key */
            /* Duplicate allowed */
            fprintf(sql_file, ",\n  KEY %s
(",quote_name(row[2],name_buff,0));
          else if (keynr == primary_key)
            fputs(",\n  PRIMARY KEY (",sql_file); /* First UNIQUE is
primary */
          else
            fprintf(sql_file, ",\n  UNIQUE %s
(",quote_name(row[2],name_buff,
                                                              0));
        }
        else
          putc(',', sql_file);
        fputs(quote_name(row[4], name_buff, 0), sql_file);
        if (row[7])
          fprintf(sql_file, " (%s)",row[7]);      /* Sub key */
        check_io(sql_file);
      }
      mysql_free_result(result);
      if (!opt_xml)
      {
        if (keynr)
          putc(')', sql_file);
        fputs("\n)",sql_file);
        check_io(sql_file);
      }

      /* Get MySQL specific create options */
      if (create_options)
      {
        char show_name_buff[NAME_LEN*2+2+24];

        /* Check memory for quote_for_like() */
        my_snprintf(buff, sizeof(buff), "show table status like %s",
                    quote_for_like(table, show_name_buff));

        if (mysql_query_with_error_report(mysql, &result, buff))
        {
          if (mysql_errno(mysql) != ER_PARSE_ERROR)
          {                                     /* If old MySQL version
*/
```

```
            verbose_msg("-- Warning: Couldn't get status information for
" \
                        "table %s (%s)\n",
result_table,mysql_error(mysql));
          }
        }
        else if (!(row= mysql_fetch_row(result)))
        {
          fprintf(stderr,
                  "Error: Couldn't read status information for table %s
(%s)\n",
                  result_table,mysql_error(mysql));
        }
        else
        {
          if (opt_xml)
            print_xml_row(sql_file, "options", result, &row, NullS);
          else
          {
            fputs("/*!",sql_file);
            print_value(sql_file,result,row,"engine=","Engine",0);
            print_value(sql_file,result,row,"","Create_options",0);
            print_value(sql_file,result,row,"comment=","Comment",1);
            fputs(" */",sql_file);
            check_io(sql_file);
          }
        }
        mysql_free_result(result);              /* Is always safe to free
*/
      }
continue_xml:
      if (!opt_xml)
        fputs(";\n", sql_file);
      else
        fputs("\t</table_structure>\n", sql_file);
      check_io(sql_file);
    }
  }
  if (complete_insert)
  {
    dynstr_append_checked(&insert_pat, ") VALUES ");
    if (!extended_insert)
      dynstr_append_checked(&insert_pat, "(");
  }
  if (sql_file != md_result_file)
  {
    fputs("\n", sql_file);
    write_footer(sql_file);
    my_fclose(sql_file, MYF(MY_WME));
  }
  DBUG_RETURN((uint) num_fields);
} /* get_table_structure */
<sep>
static inline unsigned char get_pixel_color(int n, size_t row)
```

```
{
    return (n & (1 << (nstripes - 1 - row))) ? '\xc0' : '\x40';
}
<sep>
MagickPrivate void XFileBrowserWidget(Display *display,XWindows *windows,
  const char *action,char *reply)
{
#define CancelButtonText  "Cancel"
#define DirectoryText  "Directory:"
#define FilenameText  "File name:"
#define GrabButtonText  "Grab"
#define FormatButtonText  "Format"
#define HomeButtonText  "Home"
#define UpButtonText  "Up"

  char
    *directory,
    **filelist,
    home_directory[MagickPathExtent],
    primary_selection[MagickPathExtent],
    text[MagickPathExtent],
    working_path[MagickPathExtent];

  int
    x,
    y;

  ssize_t
    i;

  static char
    glob_pattern[MagickPathExtent] = "*",
    format[MagickPathExtent] = "miff";

  static MagickStatusType
    mask = (MagickStatusType) (CWWidth | CWHeight | CWX | CWY);

  Status
    status;

  unsigned int
    anomaly,
    height,
    text_width,
    visible_files,
    width;

  size_t
    delay,
    files,
    state;

  XEvent
    event;
```

```
  XFontStruct
    *font_info;

  XTextProperty
    window_name;

  XWidgetInfo
    action_info,
    cancel_info,
    expose_info,
    special_info,
    list_info,
    home_info,
    north_info,
    reply_info,
    scroll_info,
    selection_info,
    slider_info,
    south_info,
    text_info,
    up_info;

  XWindowChanges
    window_changes;

  /*
    Read filelist from current directory.
  */
  assert(display != (Display *) NULL);
  assert(windows != (XWindows *) NULL);
  assert(action != (char *) NULL);
  assert(reply != (char *) NULL);
  (void) LogMagickEvent(TraceEvent,GetMagickModule(),"%s",action);
  XSetCursorState(display,windows,MagickTrue);
  XCheckRefreshWindows(display,windows);
  directory=getcwd(home_directory,MagickPathExtent);
  (void) directory;
  (void) CopyMagickString(working_path,home_directory,MagickPathExtent);
  filelist=ListFiles(working_path,glob_pattern,&files);
  if (filelist == (char **) NULL)
    {
      /*
        Directory read failed.
      */
      XNoticeWidget(display,windows,"Unable to read
directory:",working_path);
      (void) XDialogWidget(display,windows,action,"Enter
filename:",reply);
      return;
    }
  /*
    Determine File Browser widget attributes.
  */
```

```
  font_info=windows->widget.font_info;
  text_width=0;
  for (i=0; i < (ssize_t) files; i++)
    if (WidgetTextWidth(font_info,filelist[i]) > text_width)
      text_width=WidgetTextWidth(font_info,filelist[i]);
  width=WidgetTextWidth(font_info,(char *) action);
  if (WidgetTextWidth(font_info,GrabButtonText) > width)
    width=WidgetTextWidth(font_info,GrabButtonText);
  if (WidgetTextWidth(font_info,FormatButtonText) > width)
    width=WidgetTextWidth(font_info,FormatButtonText);
  if (WidgetTextWidth(font_info,CancelButtonText) > width)
    width=WidgetTextWidth(font_info,CancelButtonText);
  if (WidgetTextWidth(font_info,HomeButtonText) > width)
    width=WidgetTextWidth(font_info,HomeButtonText);
  if (WidgetTextWidth(font_info,UpButtonText) > width)
    width=WidgetTextWidth(font_info,UpButtonText);
  width+=QuantumMargin;
  if (WidgetTextWidth(font_info,DirectoryText) > width)
    width=WidgetTextWidth(font_info,DirectoryText);
  if (WidgetTextWidth(font_info,FilenameText) > width)
    width=WidgetTextWidth(font_info,FilenameText);
  height=(unsigned int) (font_info->ascent+font_info->descent);
  /*
    Position File Browser widget.
  */
  windows->widget.width=width+MagickMin((int) text_width,(int)
MaxTextWidth)+
    6*QuantumMargin;
  windows->widget.min_width=width+MinTextWidth+4*QuantumMargin;
  if (windows->widget.width < windows->widget.min_width)
    windows->widget.width=windows->widget.min_width;
  windows->widget.height=(unsigned int)
    (((81*height) >> 2)+((13*QuantumMargin) >> 1)+4);
  windows->widget.min_height=(unsigned int)
    (((23*height) >> 1)+((13*QuantumMargin) >> 1)+4);
  if (windows->widget.height < windows->widget.min_height)
    windows->widget.height=windows->widget.min_height;
  XConstrainWindowPosition(display,&windows->widget);
  /*
    Map File Browser widget.
  */
  (void) CopyMagickString(windows->widget.name,"Browse and Select a
File",
    MagickPathExtent);
  status=XStringListToTextProperty(&windows->widget.name,1,&window_name);
  if (status != False)
    {
      XSetWMName(display,windows->widget.id,&window_name);
      XSetWMIconName(display,windows->widget.id,&window_name);
      (void) XFree((void *) window_name.value);
    }
  window_changes.width=(int) windows->widget.width;
  window_changes.height=(int) windows->widget.height;
  window_changes.x=windows->widget.x;
```

```
  window_changes.y=windows->widget.y;
  (void) XReconfigureWMWindow(display,windows->widget.id,
    windows->widget.screen,mask,&window_changes);
  (void) XMapRaised(display,windows->widget.id);
  windows->widget.mapped=MagickFalse;
  /*
    Respond to X events.
  */
  XGetWidgetInfo((char *) NULL,&slider_info);
  XGetWidgetInfo((char *) NULL,&north_info);
  XGetWidgetInfo((char *) NULL,&south_info);
  XGetWidgetInfo((char *) NULL,&expose_info);
  visible_files=0;
  anomaly=(LocaleCompare(action,"Composite") == 0) ||
    (LocaleCompare(action,"Open") == 0) || (LocaleCompare(action,"Map")
== 0);
  *reply='\0';
  delay=SuspendTime << 2;
  state=UpdateConfigurationState;
  do
  {
    if (state & UpdateConfigurationState)
      {
        int
          id;

        /*
          Initialize button information.
        */
        XGetWidgetInfo(CancelButtonText,&cancel_info);
        cancel_info.width=width;
        cancel_info.height=(unsigned int) ((3*height) >> 1);
        cancel_info.x=(int)
          (windows->widget.width-cancel_info.width-QuantumMargin-2);
        cancel_info.y=(int)
          (windows->widget.height-cancel_info.height-QuantumMargin);
        XGetWidgetInfo(action,&action_info);
        action_info.width=width;
        action_info.height=(unsigned int) ((3*height) >> 1);
        action_info.x=cancel_info.x-(cancel_info.width+(QuantumMargin >>
1)+
          (action_info.bevel_width << 1));
        action_info.y=cancel_info.y;
        XGetWidgetInfo(GrabButtonText,&special_info);
        special_info.width=width;
        special_info.height=(unsigned int) ((3*height) >> 1);
        special_info.x=action_info.x-(action_info.width+(QuantumMargin >>
1)+
          (special_info.bevel_width << 1));
        special_info.y=action_info.y;
        if (anomaly == MagickFalse)
          {
            char
              *p;
```

```
            special_info.text=(char *) FormatButtonText;
            p=reply+Extent(reply)-1;
            while ((p > (reply+1)) && (*(p-1) != '.'))
              p--;
            if ((p > (reply+1)) && (*(p-1) == '.'))
              (void) CopyMagickString(format,p,MagickPathExtent);
          }
        XGetWidgetInfo(UpButtonText,&up_info);
        up_info.width=width;
        up_info.height=(unsigned int) ((3*height) >> 1);
        up_info.x=QuantumMargin;
        up_info.y=((5*QuantumMargin) >> 1)+height;
        XGetWidgetInfo(HomeButtonText,&home_info);
        home_info.width=width;
        home_info.height=(unsigned int) ((3*height) >> 1);
        home_info.x=QuantumMargin;
        home_info.y=up_info.y+up_info.height+QuantumMargin;
        /*
          Initialize reply information.
        */
        XGetWidgetInfo(reply,&reply_info);
        reply_info.raised=MagickFalse;
        reply_info.bevel_width--;
        reply_info.width=windows->widget.width-width-((6*QuantumMargin)
>> 1);
        reply_info.height=height << 1;
        reply_info.x=(int) (width+(QuantumMargin << 1));
        reply_info.y=action_info.y-reply_info.height-QuantumMargin;
        /*
          Initialize scroll information.
        */
        XGetWidgetInfo((char *) NULL,&scroll_info);
        scroll_info.bevel_width--;
        scroll_info.width=height;
        scroll_info.height=(unsigned int)
          (reply_info.y-up_info.y-(QuantumMargin >> 1));
        scroll_info.x=reply_info.x+(reply_info.width-scroll_info.width);
        scroll_info.y=up_info.y-reply_info.bevel_width;
        scroll_info.raised=MagickFalse;
        scroll_info.trough=MagickTrue;
        north_info=scroll_info;
        north_info.raised=MagickTrue;
        north_info.width-=(north_info.bevel_width << 1);
        north_info.height=north_info.width-1;
        north_info.x+=north_info.bevel_width;
        north_info.y+=north_info.bevel_width;
        south_info=north_info;
        south_info.y=scroll_info.y+scroll_info.height-
scroll_info.bevel_width-
          south_info.height;
        id=slider_info.id;
        slider_info=north_info;
        slider_info.id=id;
```

```
        slider_info.width-=2;

slider_info.min_y=north_info.y+north_info.height+north_info.bevel_width+
          slider_info.bevel_width+2;
        slider_info.height=scroll_info.height-((slider_info.min_y-
          scroll_info.y+1) << 1)+4;
        visible_files=scroll_info.height/(height+(height >> 3));
        if (files > visible_files)
          slider_info.height=(unsigned int)
            ((visible_files*slider_info.height)/files);
        slider_info.max_y=south_info.y-south_info.bevel_width-
          slider_info.bevel_width-2;
        slider_info.x=scroll_info.x+slider_info.bevel_width+1;
        slider_info.y=slider_info.min_y;
        expose_info=scroll_info;
        expose_info.y=slider_info.y;
        /*
          Initialize list information.
        */
        XGetWidgetInfo((char *) NULL,&list_info);
        list_info.raised=MagickFalse;
        list_info.bevel_width--;
        list_info.width=(unsigned int)
          (scroll_info.x-reply_info.x-(QuantumMargin >> 1));
        list_info.height=scroll_info.height;
        list_info.x=reply_info.x;
        list_info.y=scroll_info.y;
        if (windows->widget.mapped == MagickFalse)
          state|=JumpListState;
        /*
          Initialize text information.
        */
        *text='\0';
        XGetWidgetInfo(text,&text_info);
        text_info.center=MagickFalse;
        text_info.width=reply_info.width;
        text_info.height=height;
        text_info.x=list_info.x-(QuantumMargin >> 1);
        text_info.y=QuantumMargin;
        /*
          Initialize selection information.
        */
        XGetWidgetInfo((char *) NULL,&selection_info);
        selection_info.center=MagickFalse;
        selection_info.width=list_info.width;
        selection_info.height=(unsigned int) ((9*height) >> 3);
        selection_info.x=list_info.x;
        state&=(~UpdateConfigurationState);
      }
    if (state & RedrawWidgetState)
      {
        /*
          Redraw File Browser window.
        */
```

```
        x=QuantumMargin;
        y=text_info.y+((text_info.height-height) >> 1)+font_info->ascent;
        (void) XDrawString(display,windows->widget.id,
          windows->widget.annotate_context,x,y,DirectoryText,
          Extent(DirectoryText));
        (void)
CopyMagickString(text_info.text,working_path,MagickPathExtent);
        (void) ConcatenateMagickString(text_info.text,DirectorySeparator,
          MagickPathExtent);
        (void) ConcatenateMagickString(text_info.text,glob_pattern,
          MagickPathExtent);
        XDrawWidgetText(display,&windows->widget,&text_info);
        XDrawBeveledButton(display,&windows->widget,&up_info);
        XDrawBeveledButton(display,&windows->widget,&home_info);
        XDrawBeveledMatte(display,&windows->widget,&list_info);
        XDrawBeveledMatte(display,&windows->widget,&scroll_info);
        XDrawTriangleNorth(display,&windows->widget,&north_info);
        XDrawBeveledButton(display,&windows->widget,&slider_info);
        XDrawTriangleSouth(display,&windows->widget,&south_info);
        x=QuantumMargin;
        y=reply_info.y+((reply_info.height-height) >> 1)+font_info-
>ascent;
        (void) XDrawString(display,windows->widget.id,
          windows->widget.annotate_context,x,y,FilenameText,
          Extent(FilenameText));
        XDrawBeveledMatte(display,&windows->widget,&reply_info);
        XDrawMatteText(display,&windows->widget,&reply_info);
        XDrawBeveledButton(display,&windows->widget,&special_info);
        XDrawBeveledButton(display,&windows->widget,&action_info);
        XDrawBeveledButton(display,&windows->widget,&cancel_info);
        XHighlightWidget(display,&windows-
>widget,BorderOffset,BorderOffset);
        selection_info.id=(~0);
        state|=RedrawListState;
        state&=(~RedrawWidgetState);
      }
    if (state & UpdateListState)
      {
        char
          **checklist;

        size_t
          number_files;

        /*
          Update file list.
        */
        checklist=ListFiles(working_path,glob_pattern,&number_files);
        if (checklist == (char **) NULL)
          {
            /*
              Reply is a filename, exit.
            */
            action_info.raised=MagickFalse;
```

```
            XDrawBeveledButton(display,&windows->widget,&action_info);
            break;
          }
        for (i=0; i < (ssize_t) files; i++)
          filelist[i]=DestroyString(filelist[i]);
        if (filelist != (char **) NULL)
          filelist=(char **) RelinquishMagickMemory(filelist);
        filelist=checklist;
        files=number_files;
        /*
          Update file list.
        */
        slider_info.height=
          scroll_info.height-((slider_info.min_y-scroll_info.y+1) <<
1)+1;
        if (files > visible_files)
          slider_info.height=(unsigned int)
            ((visible_files*slider_info.height)/files);
        slider_info.max_y=south_info.y-south_info.bevel_width-
          slider_info.bevel_width-2;
        slider_info.id=0;
        slider_info.y=slider_info.min_y;
        expose_info.y=slider_info.y;
        selection_info.id=(~0);
        list_info.id=(~0);
        state|=RedrawListState;
        /*
          Redraw directory name & reply.
        */
        if (IsGlob(reply_info.text) == MagickFalse)
          {
            *reply_info.text='\0';
            reply_info.cursor=reply_info.text;
          }
        (void)
CopyMagickString(text_info.text,working_path,MagickPathExtent);
        (void) ConcatenateMagickString(text_info.text,DirectorySeparator,
          MagickPathExtent);
        (void) ConcatenateMagickString(text_info.text,glob_pattern,
          MagickPathExtent);
        XDrawWidgetText(display,&windows->widget,&text_info);
        XDrawMatteText(display,&windows->widget,&reply_info);
        XDrawBeveledMatte(display,&windows->widget,&scroll_info);
        XDrawTriangleNorth(display,&windows->widget,&north_info);
        XDrawBeveledButton(display,&windows->widget,&slider_info);
        XDrawTriangleSouth(display,&windows->widget,&south_info);
        XHighlightWidget(display,&windows-
>widget,BorderOffset,BorderOffset);
        state&=(~UpdateListState);
      }
    if (state & JumpListState)
      {
        /*
          Jump scroll to match user filename.
```

```
        */
        list_info.id=(~0);
        for (i=0; i < (ssize_t) files; i++)
          if (LocaleCompare(filelist[i],reply) >= 0)
            {
              list_info.id=(int)
                (LocaleCompare(filelist[i],reply) == 0 ? i : ~0);
              break;
            }
        if ((i < (ssize_t) slider_info.id) ||
            (i >= (ssize_t) (slider_info.id+visible_files)))
          slider_info.id=(int) i-(visible_files >> 1);
        selection_info.id=(~0);
        state|=RedrawListState;
        state&=(~JumpListState);
      }
    if (state & RedrawListState)
      {
        /*
          Determine slider id and position.
        */
        if (slider_info.id >= (int) (files-visible_files))
          slider_info.id=(int) (files-visible_files);
        if ((slider_info.id < 0) || (files <= visible_files))
          slider_info.id=0;
        slider_info.y=slider_info.min_y;
        if (files > 0)
          slider_info.y+=((ssize_t) slider_info.id*(slider_info.max_y-
            slider_info.min_y+1)/files);
        if (slider_info.id != selection_info.id)
          {
            /*
              Redraw scroll bar and file names.
            */
            selection_info.id=slider_info.id;
            selection_info.y=list_info.y+(height >> 3)+2;
            for (i=0; i < (ssize_t) visible_files; i++)
            {
              selection_info.raised=(int) (slider_info.id+i) !=
list_info.id ?
                MagickTrue : MagickFalse;
              selection_info.text=(char *) NULL;
              if ((slider_info.id+i) < (ssize_t) files)
                selection_info.text=filelist[slider_info.id+i];
              XDrawWidgetText(display,&windows->widget,&selection_info);
              selection_info.y+=(int) selection_info.height;
            }
            /*
              Update slider.
            */
            if (slider_info.y > expose_info.y)
              {
                expose_info.height=(unsigned int) slider_info.y-
expose_info.y;
```

```
                expose_info.y=slider_info.y-expose_info.height-
                  slider_info.bevel_width-1;
              }
            else
              {
                expose_info.height=(unsigned int) expose_info.y-
slider_info.y;
                expose_info.y=slider_info.y+slider_info.height+
                  slider_info.bevel_width+1;
              }
            XDrawTriangleNorth(display,&windows->widget,&north_info);
            XDrawMatte(display,&windows->widget,&expose_info);
            XDrawBeveledButton(display,&windows->widget,&slider_info);
            XDrawTriangleSouth(display,&windows->widget,&south_info);
            expose_info.y=slider_info.y;
          }
        state&=(~RedrawListState);
      }
    /*
      Wait for next event.
    */
    if (north_info.raised && south_info.raised)
      (void) XIfEvent(display,&event,XScreenEvent,(char *) windows);
    else
      {
        /*
          Brief delay before advancing scroll bar.
        */
        XDelay(display,delay);
        delay=SuspendTime;
        (void) XCheckIfEvent(display,&event,XScreenEvent,(char *)
windows);
        if (north_info.raised == MagickFalse)
          if (slider_info.id > 0)
            {
              /*
                Move slider up.
              */
              slider_info.id--;
              state|=RedrawListState;
            }
        if (south_info.raised == MagickFalse)
          if (slider_info.id < (int) files)
            {
              /*
                Move slider down.
              */
              slider_info.id++;
              state|=RedrawListState;
            }
        if (event.type != ButtonRelease)
          continue;
      }
    switch (event.type)
```

```
{
  case ButtonPress:
  {
    if (MatteIsActive(slider_info,event.xbutton))
      {
        /*
          Track slider.
        */
        slider_info.active=MagickTrue;
        break;
      }
    if (MatteIsActive(north_info,event.xbutton))
      if (slider_info.id > 0)
        {
          /*
            Move slider up.
          */
          north_info.raised=MagickFalse;
          slider_info.id--;
          state|=RedrawListState;
          break;
        }
    if (MatteIsActive(south_info,event.xbutton))
      if (slider_info.id < (int) files)
        {
          /*
            Move slider down.
          */
          south_info.raised=MagickFalse;
          slider_info.id++;
          state|=RedrawListState;
          break;
        }
    if (MatteIsActive(scroll_info,event.xbutton))
      {
        /*
          Move slider.
        */
        if (event.xbutton.y < slider_info.y)
          slider_info.id-=(visible_files-1);
        else
          slider_info.id+=(visible_files-1);
        state|=RedrawListState;
        break;
      }
    if (MatteIsActive(list_info,event.xbutton))
      {
        int
          id;

        /*
          User pressed file matte.
        */
```

```
            id=slider_info.id+(event.xbutton.y-(list_info.y+(height >>
1))+1)/
              selection_info.height;
            if (id >= (int) files)
              break;
            (void)
CopyMagickString(reply_info.text,filelist[id],MagickPathExtent);
            reply_info.highlight=MagickFalse;
            reply_info.marker=reply_info.text;
            reply_info.cursor=reply_info.text+Extent(reply_info.text);
            XDrawMatteText(display,&windows->widget,&reply_info);
            if (id == list_info.id)
              {
                char
                  *p;

                p=reply_info.text+strlen(reply_info.text)-1;
                if (*p == *DirectorySeparator)
                  ChopPathComponents(reply_info.text,1);
                (void)
ConcatenateMagickString(working_path,DirectorySeparator,
                  MagickPathExtent);
                (void)
ConcatenateMagickString(working_path,reply_info.text,
                  MagickPathExtent);
                *reply='\0';
                state|=UpdateListState;
              }
            selection_info.id=(~0);
            list_info.id=id;
            state|=RedrawListState;
            break;
          }
        if (MatteIsActive(up_info,event.xbutton))
          {
            /*
              User pressed Up button.
            */
            up_info.raised=MagickFalse;
            XDrawBeveledButton(display,&windows->widget,&up_info);
            break;
          }
        if (MatteIsActive(home_info,event.xbutton))
          {
            /*
              User pressed Home button.
            */
            home_info.raised=MagickFalse;
            XDrawBeveledButton(display,&windows->widget,&home_info);
            break;
          }
        if (MatteIsActive(special_info,event.xbutton))
          {
            /*
```

```
              User pressed Special button.
            */
            special_info.raised=MagickFalse;
            XDrawBeveledButton(display,&windows->widget,&special_info);
            break;
          }
        if (MatteIsActive(action_info,event.xbutton))
          {
            /*
              User pressed action button.
            */
            action_info.raised=MagickFalse;
            XDrawBeveledButton(display,&windows->widget,&action_info);
            break;
          }
        if (MatteIsActive(cancel_info,event.xbutton))
          {
            /*
              User pressed Cancel button.
            */
            cancel_info.raised=MagickFalse;
            XDrawBeveledButton(display,&windows->widget,&cancel_info);
            break;
          }
        if (MatteIsActive(reply_info,event.xbutton) == MagickFalse)
          break;
        if (event.xbutton.button != Button2)
          {
            static Time
              click_time;

            /*
              Move text cursor to position of button press.
            */
            x=event.xbutton.x-reply_info.x-(QuantumMargin >> 2);
            for (i=1; i <= (ssize_t) Extent(reply_info.marker); i++)
              if (XTextWidth(font_info,reply_info.marker,(int) i) > x)
                break;
            reply_info.cursor=reply_info.marker+i-1;
            if (event.xbutton.time > (click_time+DoubleClick))
              reply_info.highlight=MagickFalse;
            else
              {
                /*
                  Become the XA_PRIMARY selection owner.
                */
                (void)
CopyMagickString(primary_selection,reply_info.text,
                  MagickPathExtent);
                (void) XSetSelectionOwner(display,XA_PRIMARY,windows-
>widget.id,
                  event.xbutton.time);

reply_info.highlight=XGetSelectionOwner(display,XA_PRIMARY) ==
```

```
                windows->widget.id ? MagickTrue : MagickFalse;
            }
          XDrawMatteText(display,&windows->widget,&reply_info);
          click_time=event.xbutton.time;
          break;
        }
      /*
        Request primary selection.
      */
      (void) XConvertSelection(display,XA_PRIMARY,XA_STRING,XA_STRING,
        windows->widget.id,event.xbutton.time);
      break;
    }
    case ButtonRelease:
    {
      if (windows->widget.mapped == MagickFalse)
        break;
      if (north_info.raised == MagickFalse)
        {
          /*
            User released up button.
          */
          delay=SuspendTime << 2;
          north_info.raised=MagickTrue;
          XDrawTriangleNorth(display,&windows->widget,&north_info);
        }
      if (south_info.raised == MagickFalse)
        {
          /*
            User released down button.
          */
          delay=SuspendTime << 2;
          south_info.raised=MagickTrue;
          XDrawTriangleSouth(display,&windows->widget,&south_info);
        }
      if (slider_info.active)
        {
          /*
            Stop tracking slider.
          */
          slider_info.active=MagickFalse;
          break;
        }
      if (up_info.raised == MagickFalse)
        {
          if (event.xbutton.window == windows->widget.id)
            if (MatteIsActive(up_info,event.xbutton))
              {
                ChopPathComponents(working_path,1);
                if (*working_path == '\0')
                  (void)
CopyMagickString(working_path,DirectorySeparator,
                    MagickPathExtent);
                state|=UpdateListState;
```

```
                }
              up_info.raised=MagickTrue;
              XDrawBeveledButton(display,&windows->widget,&up_info);
            }
          if (home_info.raised == MagickFalse)
            {
              if (event.xbutton.window == windows->widget.id)
                if (MatteIsActive(home_info,event.xbutton))
                  {
                    (void) CopyMagickString(working_path,home_directory,
                      MagickPathExtent);
                    state|=UpdateListState;
                  }
              home_info.raised=MagickTrue;
              XDrawBeveledButton(display,&windows->widget,&home_info);
            }
          if (special_info.raised == MagickFalse)
            {
              if (anomaly == MagickFalse)
                {
                  char
                    **formats;

                  ExceptionInfo
                    *exception;

                  size_t
                    number_formats;

                  /*
                    Let user select image format.
                  */
                  exception=AcquireExceptionInfo();
                  formats=GetMagickList("*",&number_formats,exception);
                  exception=DestroyExceptionInfo(exception);
                  if (formats == (char **) NULL)
                    break;
                  (void) XCheckDefineCursor(display,windows->widget.id,
                    windows->widget.busy_cursor);
                  windows->popup.x=windows->widget.x+60;
                  windows->popup.y=windows->widget.y+60;
                  XListBrowserWidget(display,windows,&windows->popup,
                    (const char **) formats,"Select","Select image format
type:",
                    format);
                  XSetCursorState(display,windows,MagickTrue);
                  (void) XCheckDefineCursor(display,windows->widget.id,
                    windows->widget.cursor);
                  LocaleLower(format);
                  AppendImageFormat(format,reply_info.text);

reply_info.cursor=reply_info.text+Extent(reply_info.text);
                  XDrawMatteText(display,&windows->widget,&reply_info);
                  special_info.raised=MagickTrue;
```

```
                XDrawBeveledButton(display,&windows-
>widget,&special_info);
                for (i=0; i < (ssize_t) number_formats; i++)
                  formats[i]=DestroyString(formats[i]);
                formats=(char **) RelinquishMagickMemory(formats);
                break;
              }
            if (event.xbutton.window == windows->widget.id)
              if (MatteIsActive(special_info,event.xbutton))
                {
                  (void)
CopyMagickString(working_path,"x:",MagickPathExtent);
                  state|=ExitState;
                }
            special_info.raised=MagickTrue;
            XDrawBeveledButton(display,&windows->widget,&special_info);
          }
        if (action_info.raised == MagickFalse)
          {
            if (event.xbutton.window == windows->widget.id)
              {
                if (MatteIsActive(action_info,event.xbutton))
                  {
                    if (*reply_info.text == '\0')
                      (void) XBell(display,0);
                    else
                      state|=ExitState;
                  }
              }
            action_info.raised=MagickTrue;
            XDrawBeveledButton(display,&windows->widget,&action_info);
          }
        if (cancel_info.raised == MagickFalse)
          {
            if (event.xbutton.window == windows->widget.id)
              if (MatteIsActive(cancel_info,event.xbutton))
                {
                  *reply_info.text='\0';
                  *reply='\0';
                  state|=ExitState;
                }
            cancel_info.raised=MagickTrue;
            XDrawBeveledButton(display,&windows->widget,&cancel_info);
          }
        break;
      }
      case ClientMessage:
      {
        /*
          If client window delete message, exit.
        */
        if (event.xclient.message_type != windows->wm_protocols)
          break;
        if (*event.xclient.data.l == (int) windows->wm_take_focus)
```

```
          {
            (void)
XSetInputFocus(display,event.xclient.window,RevertToParent,
              (Time) event.xclient.data.l[1]);
            break;
          }
        if (*event.xclient.data.l != (int) windows->wm_delete_window)
          break;
        if (event.xclient.window == windows->widget.id)
          {
            *reply_info.text='\0';
            state|=ExitState;
            break;
          }
        break;
      }
      case ConfigureNotify:
      {
        /*
          Update widget configuration.
        */
        if (event.xconfigure.window != windows->widget.id)
          break;
        if ((event.xconfigure.width == (int) windows->widget.width) &&
            (event.xconfigure.height == (int) windows->widget.height))
          break;
        windows->widget.width=(unsigned int)
          MagickMax(event.xconfigure.width,(int) windows-
>widget.min_width);
        windows->widget.height=(unsigned int)
          MagickMax(event.xconfigure.height,(int) windows-
>widget.min_height);
        state|=UpdateConfigurationState;
        break;
      }
      case EnterNotify:
      {
        if (event.xcrossing.window != windows->widget.id)
          break;
        state&=(~InactiveWidgetState);
        break;
      }
      case Expose:
      {
        if (event.xexpose.window != windows->widget.id)
          break;
        if (event.xexpose.count != 0)
          break;
        state|=RedrawWidgetState;
        break;
      }
      case KeyPress:
      {
        static char
```

```
  command[MagickPathExtent];

static int
  length;

static KeySym
  key_symbol;

/*
  Respond to a user key press.
*/
if (event.xkey.window != windows->widget.id)
  break;
length=XLookupString((XKeyEvent *) &event.xkey,command,
  (int) sizeof(command),&key_symbol,(XComposeStatus *) NULL);
*(command+length)='\0';
if (AreaIsActive(scroll_info,event.xkey))
  {
    /*
      Move slider.
    */
    switch ((int) key_symbol)
    {
      case XK_Home:
      case XK_KP_Home:
      {
        slider_info.id=0;
        break;
      }
      case XK_Up:
      case XK_KP_Up:
      {
        slider_info.id--;
        break;
      }
      case XK_Down:
      case XK_KP_Down:
      {
        slider_info.id++;
        break;
      }
      case XK_Prior:
      case XK_KP_Prior:
      {
        slider_info.id-=visible_files;
        break;
      }
      case XK_Next:
      case XK_KP_Next:
      {
        slider_info.id+=visible_files;
        break;
      }
      case XK_End:
```

```
              case XK_KP_End:
              {
                slider_info.id=(int) files;
                break;
              }
            }
            state|=RedrawListState;
            break;
          }
        if ((key_symbol == XK_Return) || (key_symbol == XK_KP_Enter))
          {
            /*
              Read new directory or glob patterm.
            */
            if (*reply_info.text == '\0')
              break;
            if (IsGlob(reply_info.text))
              (void) CopyMagickString(glob_pattern,reply_info.text,
                MagickPathExtent);
            else
              {
                (void)
ConcatenateMagickString(working_path,DirectorySeparator,
                  MagickPathExtent);
                (void)
ConcatenateMagickString(working_path,reply_info.text,
                  MagickPathExtent);
                if (*working_path == '~')
                  ExpandFilename(working_path);
                *reply='\0';
              }
            state|=UpdateListState;
            break;
          }
        if (key_symbol == XK_Control_L)
          {
            state|=ControlState;
            break;
          }
        if (state & ControlState)
          switch ((int) key_symbol)
          {
            case XK_u:
            case XK_U:
            {
              /*
                Erase the entire line of text.
              */
              *reply_info.text='\0';
              reply_info.cursor=reply_info.text;
              reply_info.marker=reply_info.text;
              reply_info.highlight=MagickFalse;
              break;
            }
```

```
      default:
        break;
    }
  XEditText(display,&reply_info,key_symbol,command,state);
  XDrawMatteText(display,&windows->widget,&reply_info);
  state|=JumpListState;
  break;
}
case KeyRelease:
{
  static char
    command[MagickPathExtent];

  static KeySym
    key_symbol;

  /*
    Respond to a user key release.
  */
  if (event.xkey.window != windows->widget.id)
    break;
  (void) XLookupString((XKeyEvent *) &event.xkey,command,
    (int) sizeof(command),&key_symbol,(XComposeStatus *) NULL);
  if (key_symbol == XK_Control_L)
    state&=(~ControlState);
  break;
}
case LeaveNotify:
{
  if (event.xcrossing.window != windows->widget.id)
    break;
  state|=InactiveWidgetState;
  break;
}
case MapNotify:
{
  mask&=(~CWX);
  mask&=(~CWY);
  break;
}
case MotionNotify:
{
  /*
    Discard pending button motion events.
  */
  while (XCheckMaskEvent(display,ButtonMotionMask,&event)) ;
  if (slider_info.active)
    {
      /*
        Move slider matte.
      */
      slider_info.y=event.xmotion.y-
        ((slider_info.height+slider_info.bevel_width) >> 1)+1;
      if (slider_info.y < slider_info.min_y)
```

```
              slider_info.y=slider_info.min_y;
            if (slider_info.y > slider_info.max_y)
              slider_info.y=slider_info.max_y;
            slider_info.id=0;
            if (slider_info.y != slider_info.min_y)
              slider_info.id=(int) ((files*(slider_info.y-
slider_info.min_y+1))/
                (slider_info.max_y-slider_info.min_y+1));
            state|=RedrawListState;
            break;
          }
        if (state & InactiveWidgetState)
          break;
        if (up_info.raised == MatteIsActive(up_info,event.xmotion))
          {
            /*
              Up button status changed.
            */
            up_info.raised=!up_info.raised;
            XDrawBeveledButton(display,&windows->widget,&up_info);
            break;
          }
        if (home_info.raised == MatteIsActive(home_info,event.xmotion))
          {
            /*
              Home button status changed.
            */
            home_info.raised=!home_info.raised;
            XDrawBeveledButton(display,&windows->widget,&home_info);
            break;
          }
        if (special_info.raised ==
MatteIsActive(special_info,event.xmotion))
          {
            /*
              Grab button status changed.
            */
            special_info.raised=!special_info.raised;
            XDrawBeveledButton(display,&windows->widget,&special_info);
            break;
          }
        if (action_info.raised ==
MatteIsActive(action_info,event.xmotion))
          {
            /*
              Action button status changed.
            */
            action_info.raised=action_info.raised == MagickFalse ?
              MagickTrue : MagickFalse;
            XDrawBeveledButton(display,&windows->widget,&action_info);
            break;
          }
        if (cancel_info.raised ==
MatteIsActive(cancel_info,event.xmotion))
```

```
          {
            /*
              Cancel button status changed.
            */
            cancel_info.raised=cancel_info.raised == MagickFalse ?
              MagickTrue : MagickFalse;
            XDrawBeveledButton(display,&windows->widget,&cancel_info);
            break;
          }
        break;
      }
      case SelectionClear:
      {
        reply_info.highlight=MagickFalse;
        XDrawMatteText(display,&windows->widget,&reply_info);
        break;
      }
      case SelectionNotify:
      {
        Atom
          type;

        int
          format;

        unsigned char
          *data;

        unsigned long
          after,
          length;

        /*
          Obtain response from primary selection.
        */
        if (event.xselection.property == (Atom) None)
          break;
        status=XGetWindowProperty(display,event.xselection.requestor,
          event.xselection.property,0L,2047L,MagickTrue,XA_STRING,&type,
          &format,&length,&after,&data);
        if ((status != Success) || (type != XA_STRING) || (format == 32)
||
            (length == 0))
          break;
        if ((Extent(reply_info.text)+length) >= (MagickPathExtent-1))
          (void) XBell(display,0);
        else
          {
            /*
              Insert primary selection in reply text.
            */
            *(data+length)='\0';
            XEditText(display,&reply_info,(KeySym) XK_Insert,(char *)
data,
```

```
              state);
            XDrawMatteText(display,&windows->widget,&reply_info);
            state|=JumpListState;
            state|=RedrawActionState;
          }
        (void) XFree((void *) data);
        break;
      }
      case SelectionRequest:
      {
        XSelectionEvent
          notify;

        XSelectionRequestEvent
          *request;

        if (reply_info.highlight == MagickFalse)
          break;
        /*
          Set primary selection.
        */
        request=(&(event.xselectionrequest));
        (void) XChangeProperty(request->display,request->requestor,
          request->property,request->target,8,PropModeReplace,
          (unsigned char *) primary_selection,Extent(primary_selection));
        notify.type=SelectionNotify;
        notify.display=request->display;
        notify.requestor=request->requestor;
        notify.selection=request->selection;
        notify.target=request->target;
        notify.time=request->time;
        if (request->property == None)
          notify.property=request->target;
        else
          notify.property=request->property;
        (void) XSendEvent(request->display,request->requestor,False,0,
          (XEvent *) &notify);
      }
      default:
        break;
    }
  } while ((state & ExitState) == 0);
  XSetCursorState(display,windows,MagickFalse);
  (void) XWithdrawWindow(display,windows->widget.id,windows-
>widget.screen);
  XCheckRefreshWindows(display,windows);
  /*
    Free file list.
  */
  for (i=0; i < (ssize_t) files; i++)
    filelist[i]=DestroyString(filelist[i]);
  if (filelist != (char **) NULL)
    filelist=(char **) RelinquishMagickMemory(filelist);
  if (*reply != '\0')
```

```
    {
      (void) ConcatenateMagickString(working_path,DirectorySeparator,
        MagickPathExtent);
      (void)
ConcatenateMagickString(working_path,reply,MagickPathExtent);
    }
  (void) CopyMagickString(reply,working_path,MagickPathExtent);
  if (*reply == '~')
    ExpandFilename(reply);
}
<sep>
static char *linetoken(FILE *stream)
{
    int ch, idx;

    while ((ch = fgetc(stream)) == ' ' || ch == '\t' );

    idx = 0;
    while (ch != EOF && ch != lineterm && idx < MAX_NAME)
    {
        ident[idx++] = ch;
        ch = fgetc(stream);
    } /* while */

    ungetc(ch, stream);
    ident[idx] = 0;

    return(ident);     /* returns pointer to the token */

} /* linetoken */
<sep>
static int kvm_ioctl_create_device(struct kvm *kvm,
                           struct kvm_create_device *cd)
{
      struct kvm_device_ops *ops = NULL;
      struct kvm_device *dev;
      bool test = cd->flags & KVM_CREATE_DEVICE_TEST;
      int ret;

      if (cd->type >= ARRAY_SIZE(kvm_device_ops_table))
            return -ENODEV;

      ops = kvm_device_ops_table[cd->type];
      if (ops == NULL)
            return -ENODEV;

      if (test)
            return 0;

      dev = kzalloc(sizeof(*dev), GFP_KERNEL);
      if (!dev)
            return -ENOMEM;

      dev->ops = ops;
```

```
	dev->kvm = kvm;

	mutex_lock(&kvm->lock);
	ret = ops->create(dev, cd->type);
	if (ret < 0) {
		mutex_unlock(&kvm->lock);
		kfree(dev);
		return ret;
	}
	list_add(&dev->vm_node, &kvm->devices);
	mutex_unlock(&kvm->lock);

	if (ops->init)
		ops->init(dev);

	ret = anon_inode_getfd(ops->name, &kvm_device_fops, dev, O_RDWR |
O_CLOEXEC);
	if (ret < 0) {
		mutex_lock(&kvm->lock);
		list_del(&dev->vm_node);
		mutex_unlock(&kvm->lock);
		ops->destroy(dev);
		return ret;
	}

	kvm_get_kvm(kvm);
	cd->fd = ret;
	return 0;
}
<sep>
hugetlb_get_unmapped_area(struct file *file, unsigned long addr,
		unsigned long len, unsigned long pgoff, unsigned long flags)
{
	struct hstate *h = hstate_file(file);
	struct mm_struct *mm = current->mm;
	struct vm_area_struct *vma;
	unsigned long task_size = TASK_SIZE;

	if (test_thread_flag(TIF_32BIT))
		task_size = STACK_TOP32;

	if (len & ~huge_page_mask(h))
		return -EINVAL;
	if (len > task_size)
		return -ENOMEM;

	if (flags & MAP_FIXED) {
		if (prepare_hugepage_range(file, addr, len))
			return -EINVAL;
		return addr;
	}

	if (addr) {
		addr = ALIGN(addr, huge_page_size(h));
```

```
            vma = find_vma(mm, addr);
            if (task_size - len >= addr &&
                (!vma || addr + len <= vma->vm_start))
                  return addr;
      }
      if (mm->get_unmapped_area == arch_get_unmapped_area)
            return hugetlb_get_unmapped_area_bottomup(file, addr, len,
                        pgoff, flags);
      else
            return hugetlb_get_unmapped_area_topdown(file, addr, len,
                        pgoff, flags);
}
<sep>
date_s_xmlschema(int argc, VALUE *argv, VALUE klass)
{
    VALUE str, sg;

    rb_scan_args(argc, argv, "02", &str, &sg);

    switch (argc) {
      case 0:
      str = rb_str_new2("-4712-01-01");
      case 1:
      sg = INT2FIX(DEFAULT_SG);
    }

    {
      VALUE hash = date_s__xmlschema(klass, str);
      return d_new_by_frags(klass, hash, sg);
    }
}
<sep>
htmlParseComment(htmlParserCtxtPtr ctxt) {
    xmlChar *buf = NULL;
    int len;
    int size = HTML_PARSER_BUFFER_SIZE;
    int q, ql;
    int r, rl;
    int cur, l;
    xmlParserInputState state;

    /*
     * Check that there is a comment right here.
     */
    if ((RAW != '<') || (NXT(1) != '!') ||
        (NXT(2) != '-') || (NXT(3) != '-')) return;

    state = ctxt->instate;
    ctxt->instate = XML_PARSER_COMMENT;
    SHRINK;
    SKIP(4);
    buf = (xmlChar *) xmlMallocAtomic(size * sizeof(xmlChar));
    if (buf == NULL) {
        htmlErrMemory(ctxt, "buffer allocation failed\n");
```

```
      ctxt->instate = state;
      return;
    }
    q = CUR_CHAR(ql);
    NEXTL(ql);
    r = CUR_CHAR(rl);
    NEXTL(rl);
    cur = CUR_CHAR(l);
    len = 0;
    while (IS_CHAR(cur) &&
           ((cur != '>') ||
            (r != '-') || (q != '-'))) {
      if (len + 5 >= size) {
          xmlChar *tmp;

          size *= 2;
          tmp = (xmlChar *) xmlRealloc(buf, size * sizeof(xmlChar));
          if (tmp == NULL) {
              xmlFree(buf);
              htmlErrMemory(ctxt, "growing buffer failed\n");
            ctxt->instate = state;
            return;
          }
          buf = tmp;
      }
      COPY_BUF(ql,buf,len,q);
      q = r;
      ql = rl;
      r = cur;
      rl = l;
      NEXTL(l);
      cur = CUR_CHAR(l);
      if (cur == 0) {
          SHRINK;
          GROW;
          cur = CUR_CHAR(l);
      }
    }
    buf[len] = 0;
    if (!IS_CHAR(cur)) {
      htmlParseErr(ctxt, XML_ERR_COMMENT_NOT_FINISHED,
                   "Comment not terminated \n<!--%.50s\n", buf, NULL);
      xmlFree(buf);
    } else {
        NEXT;
      if ((ctxt->sax != NULL) && (ctxt->sax->comment != NULL) &&
          (!ctxt->disableSAX))
          ctxt->sax->comment(ctxt->userData, buf);
      xmlFree(buf);
    }
    ctxt->instate = state;
}
<sep>
```

```
int phar_verify_signature(php_stream *fp, size_t end_of_phar, php_uint32
sig_type, char *sig, int sig_len, char *fname, char **signature, int
*signature_len, char **error) /* {{{ */
{
      int read_size, len;
      zend_off_t read_len;
      unsigned char buf[1024];

      php_stream_rewind(fp);

      switch (sig_type) {
            case PHAR_SIG_OPENSSL: {
#ifdef PHAR_HAVE_OPENSSL
                  BIO *in;
                  EVP_PKEY *key;
                  EVP_MD *mdtype = (EVP_MD *) EVP_sha1();
                  EVP_MD_CTX md_ctx;
#else
                  int tempsig;
#endif
                  zend_string *pubkey = NULL;
                  char *pfile;
                  php_stream *pfp;
#ifndef PHAR_HAVE_OPENSSL
                  if (!zend_hash_str_exists(&module_registry, "openssl",
sizeof("openssl")-1)) {
                        if (error) {
                              spprintf(error, 0, "openssl not loaded");
                        }
                        return FAILURE;
                  }
#endif
                  /* use __FILE__ . '.pubkey' for public key file */
                  spprintf(&pfile, 0, "%s.pubkey", fname);
                  pfp = php_stream_open_wrapper(pfile, "rb", 0, NULL);
                  efree(pfile);

                  if (!pfp || !(pubkey = php_stream_copy_to_mem(pfp,
PHP_STREAM_COPY_ALL, 0)) || !ZSTR_LEN(pubkey)) {
                        if (pfp) {
                              php_stream_close(pfp);
                        }
                        if (error) {
                              spprintf(error, 0, "openssl public key could
not be read");
                        }
                        return FAILURE;
                  }

                  php_stream_close(pfp);
#ifndef PHAR_HAVE_OPENSSL
                  tempsig = sig_len;
```

```
                if (FAILURE == phar_call_openssl_signverify(0, fp,
end_of_phar, pubkey ? ZSTR_VAL(pubkey) : NULL, pubkey ? ZSTR_LEN(pubkey)
: 0, &sig, &tempsig)) {
                        if (pubkey) {
                              zend_string_release(pubkey);
                        }

                        if (error) {
                              spprintf(error, 0, "openssl signature could
not be verified");
                        }

                        return FAILURE;
                }

                if (pubkey) {
                      zend_string_release(pubkey);
                }

                sig_len = tempsig;
#else
                in = BIO_new_mem_buf(pubkey ? ZSTR_VAL(pubkey) : NULL,
pubkey ? ZSTR_LEN(pubkey) : 0);

                if (NULL == in) {
                      zend_string_release(pubkey);
                      if (error) {
                            spprintf(error, 0, "openssl signature could
not be processed");
                      }
                      return FAILURE;
                }

                key = PEM_read_bio_PUBKEY(in, NULL,NULL, NULL);
                BIO_free(in);
                zend_string_release(pubkey);

                if (NULL == key) {
                      if (error) {
                            spprintf(error, 0, "openssl signature could
not be processed");
                      }
                      return FAILURE;
                }

                EVP_VerifyInit(&md_ctx, mdtype);
                read_len = end_of_phar;

                if (read_len > sizeof(buf)) {
                      read_size = sizeof(buf);
                } else {
                      read_size = (int)read_len;
                }
```

```
                php_stream_seek(fp, 0, SEEK_SET);

                while (read_size && (len = php_stream_read(fp,
(char*)buf, read_size)) > 0) {
                      EVP_VerifyUpdate (&md_ctx, buf, len);
                      read_len -= (zend_off_t)len;

                      if (read_len < read_size) {
                            read_size = (int)read_len;
                      }
                }

                if (EVP_VerifyFinal(&md_ctx, (unsigned char *)sig,
sig_len, key) != 1) {
                      /* 1: signature verified, 0: signature does not
match, -1: failed signature operation */
                      EVP_MD_CTX_cleanup(&md_ctx);

                      if (error) {
                            spprintf(error, 0, "broken openssl
signature");
                      }

                      return FAILURE;
                }

                EVP_MD_CTX_cleanup(&md_ctx);
#endif

                *signature_len = phar_hex_str((const char*)sig, sig_len,
signature);
           }
           break;
#ifdef PHAR_HASH_OK
           case PHAR_SIG_SHA512: {
                unsigned char digest[64];
                PHP_SHA512_CTX context;

                PHP_SHA512Init(&context);
                read_len = end_of_phar;

                if (read_len > sizeof(buf)) {
                      read_size = sizeof(buf);
                } else {
                      read_size = (int)read_len;
                }

                while ((len = php_stream_read(fp, (char*)buf,
read_size)) > 0) {
                      PHP_SHA512Update(&context, buf, len);
                      read_len -= (zend_off_t)len;
                      if (read_len < read_size) {
                            read_size = (int)read_len;
                      }
```

```
                }

                PHP_SHA512Final(digest, &context);

                if (memcmp(digest, sig, sizeof(digest))) {
                        if (error) {
                                spprintf(error, 0, "broken signature");
                        }
                        return FAILURE;
                }

                *signature_len = phar_hex_str((const char*)digest,
sizeof(digest), signature);
                break;
        }
        case PHAR_SIG_SHA256: {
                unsigned char digest[32];
                PHP_SHA256_CTX context;

                PHP_SHA256Init(&context);
                read_len = end_of_phar;

                if (read_len > sizeof(buf)) {
                        read_size = sizeof(buf);
                } else {
                        read_size = (int)read_len;
                }

                while ((len = php_stream_read(fp, (char*)buf,
read_size)) > 0) {
                        PHP_SHA256Update(&context, buf, len);
                        read_len -= (zend_off_t)len;
                        if (read_len < read_size) {
                                read_size = (int)read_len;
                        }
                }

                PHP_SHA256Final(digest, &context);

                if (memcmp(digest, sig, sizeof(digest))) {
                        if (error) {
                                spprintf(error, 0, "broken signature");
                        }
                        return FAILURE;
                }

                *signature_len = phar_hex_str((const char*)digest,
sizeof(digest), signature);
                break;
        }
#else
        case PHAR_SIG_SHA512:
        case PHAR_SIG_SHA256:
                if (error) {
```

```
                        spprintf(error, 0, "unsupported signature");
                    }
                    return FAILURE;
#endif
                case PHAR_SIG_SHA1: {
                    unsigned char digest[20];
                    PHP_SHA1_CTX  context;

                    PHP_SHA1Init(&context);
                    read_len = end_of_phar;

                    if (read_len > sizeof(buf)) {
                        read_size = sizeof(buf);
                    } else {
                        read_size = (int)read_len;
                    }

                    while ((len = php_stream_read(fp, (char*)buf,
read_size)) > 0) {
                        PHP_SHA1Update(&context, buf, len);
                        read_len -= (zend_off_t)len;
                        if (read_len < read_size) {
                            read_size = (int)read_len;
                        }
                    }

                    PHP_SHA1Final(digest, &context);

                    if (memcmp(digest, sig, sizeof(digest))) {
                        if (error) {
                            spprintf(error, 0, "broken signature");
                        }
                        return FAILURE;
                    }

                    *signature_len = phar_hex_str((const char*)digest,
sizeof(digest), signature);
                    break;
                }
                case PHAR_SIG_MD5: {
                    unsigned char digest[16];
                    PHP_MD5_CTX   context;

                    PHP_MD5Init(&context);
                    read_len = end_of_phar;

                    if (read_len > sizeof(buf)) {
                        read_size = sizeof(buf);
                    } else {
                        read_size = (int)read_len;
                    }

                    while ((len = php_stream_read(fp, (char*)buf,
read_size)) > 0) {
```

```
                        PHP_MD5Update(&context, buf, len);
                        read_len -= (zend_off_t)len;
                        if (read_len < read_size) {
                              read_size = (int)read_len;
                        }
                  }

                  PHP_MD5Final(digest, &context);

                  if (memcmp(digest, sig, sizeof(digest))) {
                        if (error) {
                              spprintf(error, 0, "broken signature");
                        }
                        return FAILURE;
                  }

                  *signature_len = phar_hex_str((const char*)digest,
sizeof(digest), signature);
                  break;
            }
            default:
                  if (error) {
                        spprintf(error, 0, "broken or unsupported
signature");
                  }
                  return FAILURE;
      }
      return SUCCESS;
}
<sep>
static inline void add_offset_pair(zval *result, char *str, int len, int
offset, char *name)
{
      zval *match_pair;

      ALLOC_ZVAL(match_pair);
      array_init(match_pair);
      INIT_PZVAL(match_pair);

      /* Add (match, offset) to the return value */
      add_next_index_stringl(match_pair, str, len, 1);
      add_next_index_long(match_pair, offset);

      if (name) {
            zval_add_ref(&match_pair);
            zend_hash_update(Z_ARRVAL_P(result), name, strlen(name)+1,
&match_pair, sizeof(zval *), NULL);
      }
      zend_hash_next_index_insert(Z_ARRVAL_P(result), &match_pair,
sizeof(zval *), NULL);
}
<sep>
sha1_handle_first_client_response (DBusAuth         *auth,
                                   const DBusString *data)
```

```
{
  /* We haven't sent a challenge yet, we're expecting a desired
   * username from the client.
   */
  DBusString tmp;
  DBusString tmp2;
  dbus_bool_t retval = FALSE;
  DBusError error = DBUS_ERROR_INIT;

  _dbus_string_set_length (&auth->challenge, 0);

  if (_dbus_string_get_length (data) > 0)
    {
      if (_dbus_string_get_length (&auth->identity) > 0)
        {
          /* Tried to send two auth identities, wtf */
          _dbus_verbose ("%s: client tried to send auth identity, but we
already have one\n",
                         DBUS_AUTH_NAME (auth));
          return send_rejected (auth);
        }
      else
        {
          /* this is our auth identity */
          if (!_dbus_string_copy (data, 0, &auth->identity, 0))
            return FALSE;
        }
    }

  if (!_dbus_credentials_add_from_user (auth->desired_identity, data))
    {
      _dbus_verbose ("%s: Did not get a valid username from client\n",
                     DBUS_AUTH_NAME (auth));
      return send_rejected (auth);
    }

  if (!_dbus_string_init (&tmp))
    return FALSE;

  if (!_dbus_string_init (&tmp2))
    {
      _dbus_string_free (&tmp);
      return FALSE;
    }

  /* we cache the keyring for speed, so here we drop it if it's the
   * wrong one. FIXME caching the keyring here is useless since we use
   * a different DBusAuth for every connection.
   */
  if (auth->keyring &&
      !_dbus_keyring_is_for_credentials (auth->keyring,
                                         auth->desired_identity))
    {
      _dbus_keyring_unref (auth->keyring);
```

```
      auth->keyring = NULL;
    }

  if (auth->keyring == NULL)
    {
      auth->keyring = _dbus_keyring_new_for_credentials (auth-
>desired_identity,
                                                         &auth->context,
                                                         &error);

      if (auth->keyring == NULL)
        {
          if (dbus_error_has_name (&error,
                                   DBUS_ERROR_NO_MEMORY))
            {
              dbus_error_free (&error);
              goto out;
            }
          else
            {
              _DBUS_ASSERT_ERROR_IS_SET (&error);
              _dbus_verbose ("%s: Error loading keyring: %s\n",
                             DBUS_AUTH_NAME (auth), error.message);
              if (send_rejected (auth))
                retval = TRUE; /* retval is only about mem */
              dbus_error_free (&error);
              goto out;
            }
        }
      else
        {
          _dbus_assert (!dbus_error_is_set (&error));
        }
    }

  _dbus_assert (auth->keyring != NULL);

  auth->cookie_id = _dbus_keyring_get_best_key (auth->keyring, &error);
  if (auth->cookie_id < 0)
    {
      _DBUS_ASSERT_ERROR_IS_SET (&error);
      _dbus_verbose ("%s: Could not get a cookie ID to send to client:
%s\n",
                     DBUS_AUTH_NAME (auth), error.message);
      if (send_rejected (auth))
        retval = TRUE;
      dbus_error_free (&error);
      goto out;
    }
  else
    {
      _dbus_assert (!dbus_error_is_set (&error));
    }
```

```
 if (!_dbus_string_copy (&auth->context, 0,
                         &tmp2, _dbus_string_get_length (&tmp2)))
   goto out;

 if (!_dbus_string_append (&tmp2, " "))
   goto out;

 if (!_dbus_string_append_int (&tmp2, auth->cookie_id))
   goto out;

 if (!_dbus_string_append (&tmp2, " "))
   goto out;

 if (!_dbus_generate_random_bytes (&tmp, N_CHALLENGE_BYTES, &error))
   {
     if (dbus_error_has_name (&error, DBUS_ERROR_NO_MEMORY))
       {
         dbus_error_free (&error);
         goto out;
       }
     else
       {
         _DBUS_ASSERT_ERROR_IS_SET (&error);
         _dbus_verbose ("%s: Error generating challenge: %s\n",
                        DBUS_AUTH_NAME (auth), error.message);
         if (send_rejected (auth))
           retval = TRUE; /* retval is only about mem */

         dbus_error_free (&error);
         goto out;
       }
   }

 _dbus_string_set_length (&auth->challenge, 0);
 if (!_dbus_string_hex_encode (&tmp, 0, &auth->challenge, 0))
   goto out;

 if (!_dbus_string_hex_encode (&tmp, 0, &tmp2,
                               _dbus_string_get_length (&tmp2)))
   goto out;

 if (!send_data (auth, &tmp2))
   goto out;

 goto_state (auth, &server_state_waiting_for_data);
 retval = TRUE;

out:
 _dbus_string_zero (&tmp);
 _dbus_string_free (&tmp);
 _dbus_string_zero (&tmp2);
 _dbus_string_free (&tmp2);

 return retval;
```

```
}
<sep>
static int treo_attach(struct usb_serial *serial)
{
      struct usb_serial_port *swap_port;

      /* Only do this endpoint hack for the Handspring devices with
       * interrupt in endpoints, which for now are the Treo devices. */
      if (!((le16_to_cpu(serial->dev->descriptor.idVendor)
                                  == HANDSPRING_VENDOR_ID) ||
            (le16_to_cpu(serial->dev->descriptor.idVendor)
                                  == KYOCERA_VENDOR_ID)) ||
            (serial->num_interrupt_in == 0))
            return 0;

      /*
      * It appears that Treos and Kyoceras want to use the
      * 1st bulk in endpoint to communicate with the 2nd bulk out
endpoint,
      * so let's swap the 1st and 2nd bulk in and interrupt endpoints.
      * Note that swapping the bulk out endpoints would break lots of
      * apps that want to communicate on the second port.
      */
#define COPY_PORT(dest, src)                                    \
      do { \
            int i;                                              \
                                                        \
            for (i = 0; i < ARRAY_SIZE(src->read_urbs); ++i) {  \
                  dest->read_urbs[i] = src->read_urbs[i];       \
                  dest->read_urbs[i]->context = dest;           \
                  dest->bulk_in_buffers[i] = src->bulk_in_buffers[i]; \
            }                                                   \
            dest->read_urb = src->read_urb;                     \
            dest->bulk_in_endpointAddress = src-
>bulk_in_endpointAddress;\
            dest->bulk_in_buffer = src->bulk_in_buffer;         \
            dest->bulk_in_size = src->bulk_in_size;             \
            dest->interrupt_in_urb = src->interrupt_in_urb;           \
            dest->interrupt_in_urb->context = dest;             \
            dest->interrupt_in_endpointAddress = \
                              src->interrupt_in_endpointAddress;\
            dest->interrupt_in_buffer = src->interrupt_in_buffer;     \
      } while (0);

      swap_port = kmalloc(sizeof(*swap_port), GFP_KERNEL);
      if (!swap_port)
            return -ENOMEM;
      COPY_PORT(swap_port, serial->port[0]);
      COPY_PORT(serial->port[0], serial->port[1]);
      COPY_PORT(serial->port[1], swap_port);
      kfree(swap_port);

      return 0;
}
```

```
<sep>
static int decode_tree_entry(struct tree_desc *desc, const char *buf,
unsigned long size, struct strbuf *err)
{
	const char *path;
	unsigned int mode, len;

	if (size < 23 || buf[size - 21]) {
		strbuf_addstr(err, _("too-short tree object"));
		return -1;
	}

	path = get_mode(buf, &mode);
	if (!path) {
		strbuf_addstr(err, _("malformed mode in tree entry"));
		return -1;
	}
	if (!*path) {
		strbuf_addstr(err, _("empty filename in tree entry"));
		return -1;
	}
	len = strlen(path) + 1;

	/* Initialize the descriptor entry */
	desc->entry.path = path;
	desc->entry.mode = canon_mode(mode);
	desc->entry.oid  = (const struct object_id *)(path + len);

	return 0;
}
<sep>
static int chip_write(struct CHIPSTATE *chip, int subaddr, int val)
{
	unsigned char buffer[2];

	if (-1 == subaddr) {
		v4l_dbg(1, debug, chip->c, "%s: chip_write: 0x%x\n",
			chip->c->name, val);
		chip->shadow.bytes[1] = val;
		buffer[0] = val;
		if (1 != i2c_master_send(chip->c,buffer,1)) {
			v4l_warn(chip->c, "%s: I/O error (write 0x%x)\n",
				chip->c->name, val);
			return -1;
		}
	} else {
		v4l_dbg(1, debug, chip->c, "%s: chip_write: reg%d=0x%x\n",
			chip->c->name, subaddr, val);
		chip->shadow.bytes[subaddr+1] = val;
		buffer[0] = subaddr;
		buffer[1] = val;
		if (2 != i2c_master_send(chip->c,buffer,2)) {
			v4l_warn(chip->c, "%s: I/O error (write reg%d=0x%x)\n",
			chip->c->name, subaddr, val);
```

```
                return -1;
            }
        }
        return 0;
}
<sep>
int passwd_to_utf16(unsigned char *in_passwd,
                    int length,
                    int max_length,
                    unsigned char *out_passwd)
{
#ifdef WIN32
    int ret;
    (void)length;
    ret = MultiByteToWideChar(
        CP_ACP,
        0,
        (LPCSTR)in_passwd,
        -1,
        (LPWSTR)out_passwd,
        max_length / 2
    );
    if (ret == 0)
        return AESCRYPT_READPWD_ICONV;
    return ret * 2;
#else
#ifndef ENABLE_ICONV
    /* support only latin */
    int i;
    for (i=0;i<length+1;i++) {
        out_passwd[i*2] = in_passwd[i];
        out_passwd[i*2+1] = 0;
    }
    return length*2;
#else
    unsigned char *ic_outbuf,
                  *ic_inbuf;
    iconv_t condesc;
    size_t ic_inbytesleft,
           ic_outbytesleft;

    /* Max length is specified in character, but this function deals
     * with bytes.  So, multiply by two since we are going to create a
     * UTF-16 string.
     */
    max_length *= 2;

    ic_inbuf = in_passwd;
    ic_inbytesleft = length;
    ic_outbytesleft = max_length;
    ic_outbuf = out_passwd;

    /* Set the locale based on the current environment */
    setlocale(LC_CTYPE,"");
```

```
    if ((condesc = iconv_open("UTF-16LE", nl_langinfo(CODESET))) ==
        (iconv_t)(-1))
    {
        perror("Error in iconv_open");
        return -1;
    }

    if (iconv(condesc,
              (char ** const) &ic_inbuf,
              &ic_inbytesleft,
              (char ** const) &ic_outbuf,
              &ic_outbytesleft) == (size_t) -1)
    {
        switch (errno)
        {
            case E2BIG:
                fprintf(stderr, "Error: password too long\n");
                iconv_close(condesc);
                return -1;
                break;
            default:
                /*
                printf("\nEILSEQ(%d), EINVAL(%d), %d\n",
                       EILSEQ,
                       EINVAL,
                       errno);
                */
                perror("Password conversion error");
                iconv_close(condesc);
                return -1;
        }
    }
    iconv_close(condesc);
    return (max_length - ic_outbytesleft);
#endif
#endif
}
<sep>
static void prepare_attr_stack(const char *path, int dirlen)
{
      struct attr_stack *elem, *info;
      int len;
      char pathbuf[PATH_MAX];

      /*
       * At the bottom of the attribute stack is the built-in
       * set of attribute definitions.  Then, contents from
       * .gitattribute files from directories closer to the
       * root to the ones in deeper directories are pushed
       * to the stack.  Finally, at the very top of the stack
       * we always keep the contents of $GIT_DIR/info/attributes.
       *
       * When checking, we use entries from near the top of the
```

```
 * stack, preferring $GIT_DIR/info/attributes, then
 * .gitattributes in deeper directories to shallower ones,
 * and finally use the built-in set as the default.
 */
if (!attr_stack)
      bootstrap_attr_stack();

/*
 * Pop the "info" one that is always at the top of the stack.
 */
info = attr_stack;
attr_stack = info->prev;

/*
 * Pop the ones from directories that are not the prefix of
 * the path we are checking.
 */
while (attr_stack && attr_stack->origin) {
      int namelen = strlen(attr_stack->origin);

      elem = attr_stack;
      if (namelen <= dirlen &&
          !strncmp(elem->origin, path, namelen))
            break;

      debug_pop(elem);
      attr_stack = elem->prev;
      free_attr_elem(elem);
}

/*
 * Read from parent directories and push them down
 */
if (!is_bare_repository()) {
      while (1) {
            char *cp;

            len = strlen(attr_stack->origin);
            if (dirlen <= len)
                  break;
            memcpy(pathbuf, path, dirlen);
            memcpy(pathbuf + dirlen, "/", 2);
            cp = strchr(pathbuf + len + 1, '/');
            strcpy(cp + 1, GITATTRIBUTES_FILE);
            elem = read_attr(pathbuf, 0);
            *cp = '\0';
            elem->origin = strdup(pathbuf);
            elem->prev = attr_stack;
            attr_stack = elem;
            debug_push(elem);
      }
}

/*
```

```
       * Finally push the "info" one at the top of the stack.
       */
      info->prev = attr_stack;
      attr_stack = info;
}
<sep>
ssh_scp ssh_scp_new(ssh_session session, int mode, const char *location)
{
    ssh_scp scp = NULL;

    if (session == NULL) {
        goto error;
    }

    scp = (ssh_scp)calloc(1, sizeof(struct ssh_scp_struct));
    if (scp == NULL) {
        ssh_set_error(session, SSH_FATAL,
                      "Error allocating memory for ssh_scp");
        goto error;
    }

    if ((mode & ~SSH_SCP_RECURSIVE) != SSH_SCP_WRITE &&
        (mode & ~SSH_SCP_RECURSIVE) != SSH_SCP_READ)
    {
        ssh_set_error(session, SSH_FATAL,
                      "Invalid mode %d for ssh_scp_new()", mode);
        goto error;
    }

    scp->location = strdup(location);
    if (scp->location == NULL) {
        ssh_set_error(session, SSH_FATAL,
                      "Error allocating memory for ssh_scp");
        goto error;
    }

    scp->session = session;
    scp->mode = mode & ~SSH_SCP_RECURSIVE;
    scp->recursive = (mode & SSH_SCP_RECURSIVE) != 0;
    scp->channel = NULL;
    scp->state = SSH_SCP_NEW;

    return scp;

error:
    ssh_scp_free(scp);
    return NULL;
}
<sep>
static Image *ReadXWDImage(const ImageInfo *image_info,ExceptionInfo
*exception)
{
#define CheckOverflowException(length,width,height) \
  (((height) != 0) && ((length)/((size_t) height) != ((size_t) width)))
```

```
char
  *comment;

Image
  *image;

IndexPacket
  index;

int
  x_status;

MagickBooleanType
  authentic_colormap;

MagickStatusType
  status;

register IndexPacket
  *indexes;

register ssize_t
  x;

register PixelPacket
  *q;

register ssize_t
  i;

register size_t
  pixel;

size_t
  length;

ssize_t
  count,
  y;

unsigned long
  lsb_first;

XColor
  *colors;

XImage
  *ximage;

XWDFileHeader
  header;

/*
```

```
  Open image file.
*/
assert(image_info != (const ImageInfo *) NULL);
assert(image_info->signature == MagickCoreSignature);
if (image_info->debug != MagickFalse)
  (void) LogMagickEvent(TraceEvent,GetMagickModule(),"%s",
    image_info->filename);
assert(exception != (ExceptionInfo *) NULL);
assert(exception->signature == MagickCoreSignature);
image=AcquireImage(image_info);
status=OpenBlob(image_info,image,ReadBinaryBlobMode,exception);
if (status == MagickFalse)
  {
    image=DestroyImageList(image);
    return((Image *) NULL);
  }
/*
   Read in header information.
*/
count=ReadBlob(image,sz_XWDheader,(unsigned char *) &header);
if (count != sz_XWDheader)
  ThrowReaderException(CorruptImageError,"UnableToReadImageHeader");
/*
  Ensure the header byte-order is most-significant byte first.
*/
lsb_first=1;
if ((int) (*(char *) &lsb_first) != 0)
  MSBOrderLong((unsigned char *) &header,sz_XWDheader);
/*
  Check to see if the dump file is in the proper format.
*/
if (header.file_version != XWD_FILE_VERSION)
  ThrowReaderException(CorruptImageError,"FileFormatVersionMismatch");
if (header.header_size < sz_XWDheader)
  ThrowReaderException(CorruptImageError,"ImproperImageHeader");
switch (header.visual_class)
{
  case StaticGray:
  case GrayScale:
  {
    if (header.bits_per_pixel != 1)
      ThrowReaderException(CorruptImageError,"ImproperImageHeader");
    break;
  }
  case StaticColor:
  case PseudoColor:
  {
    if ((header.bits_per_pixel < 1) || (header.bits_per_pixel > 15) ||
        (header.ncolors == 0))
      ThrowReaderException(CorruptImageError,"ImproperImageHeader");
    break;
  }
  case TrueColor:
  case DirectColor:
```

```
    {
      if ((header.bits_per_pixel != 16) && (header.bits_per_pixel != 24)
&&
          (header.bits_per_pixel != 32))
        ThrowReaderException(CorruptImageError,"ImproperImageHeader");
      break;
    }
    default:
      ThrowReaderException(CorruptImageError,"ImproperImageHeader");
  }
  switch (header.pixmap_format)
  {
    case XYBitmap:
    {
      if (header.pixmap_depth != 1)
        ThrowReaderException(CorruptImageError,"ImproperImageHeader");
      break;
    }
    case XYPixmap:
    case ZPixmap:
    {
      if ((header.pixmap_depth < 1) || (header.pixmap_depth > 32))
        ThrowReaderException(CorruptImageError,"ImproperImageHeader");
      switch (header.bitmap_pad)
      {
        case 8:
        case 16:
        case 32:
          break;
        default:
          ThrowReaderException(CorruptImageError,"ImproperImageHeader");
      }
      break;
    }
    default:
      ThrowReaderException(CorruptImageError,"ImproperImageHeader");
  }
  switch (header.bitmap_unit)
  {
    case 8:
    case 16:
    case 32:
      break;
    default:
      ThrowReaderException(CorruptImageError,"ImproperImageHeader");
  }
  switch (header.byte_order)
  {
    case LSBFirst:
    case MSBFirst:
      break;
    default:
      ThrowReaderException(CorruptImageError,"ImproperImageHeader");
  }
```

```
  switch (header.bitmap_bit_order)
  {
    case LSBFirst:
    case MSBFirst:
      break;
    default:
      ThrowReaderException(CorruptImageError,"ImproperImageHeader");
  }
  if (((header.bitmap_pad % 8) != 0) || (header.bitmap_pad > 32))
    ThrowReaderException(CorruptImageError,"ImproperImageHeader");
  if (header.ncolors > 65535)
    ThrowReaderException(CorruptImageError,"ImproperImageHeader");
  length=(size_t) (header.header_size-sz_XWDheader);
  if ((length+1) != ((size_t) ((CARD32) (length+1))))
    ThrowReaderException(CorruptImageError,"ImproperImageHeader");
  comment=(char *) AcquireQuantumMemory(length+1,sizeof(*comment));
  if (comment == (char *) NULL)
    ThrowReaderException(ResourceLimitError,"MemoryAllocationFailed");
  count=ReadBlob(image,length,(unsigned char *) comment);
  comment[length]='\0';
  (void) SetImageProperty(image,"comment",comment);
  comment=DestroyString(comment);
  if (count != (ssize_t) length)
    ThrowReaderException(CorruptImageError,"UnexpectedEndOfFile");
  /*
    Initialize the X image.
  */
  ximage=(XImage *) AcquireMagickMemory(sizeof(*ximage));
  if (ximage == (XImage *) NULL)
    ThrowReaderException(ResourceLimitError,"MemoryAllocationFailed");
  ximage->depth=(int) header.pixmap_depth;
  ximage->format=(int) header.pixmap_format;
  ximage->xoffset=(int) header.xoffset;
  ximage->data=(char *) NULL;
  ximage->width=(int) header.pixmap_width;
  ximage->height=(int) header.pixmap_height;
  ximage->bitmap_pad=(int) header.bitmap_pad;
  ximage->bytes_per_line=(int) header.bytes_per_line;
  ximage->byte_order=(int) header.byte_order;
  ximage->bitmap_unit=(int) header.bitmap_unit;
  ximage->bitmap_bit_order=(int) header.bitmap_bit_order;
  ximage->bits_per_pixel=(int) header.bits_per_pixel;
  ximage->red_mask=header.red_mask;
  ximage->green_mask=header.green_mask;
  ximage->blue_mask=header.blue_mask;
  if ((ximage->width < 0) || (ximage->height < 0) || (ximage->depth < 0)
||
      (ximage->format < 0) || (ximage->byte_order < 0) ||
      (ximage->bitmap_bit_order < 0) || (ximage->bitmap_pad < 0) ||
      (ximage->bytes_per_line < 0))
    {
      ximage=(XImage *) RelinquishMagickMemory(ximage);
      ThrowReaderException(CorruptImageError,"ImproperImageHeader");
    }
```

```
  if ((ximage->width > 65535) || (ximage->height > 65535))
    {
      ximage=(XImage *) RelinquishMagickMemory(ximage);
      ThrowReaderException(CorruptImageError,"ImproperImageHeader");
    }
  if ((ximage->bits_per_pixel > 32) || (ximage->bitmap_unit > 32))
    {
      ximage=(XImage *) RelinquishMagickMemory(ximage);
      ThrowReaderException(CorruptImageError,"ImproperImageHeader");
    }
  x_status=XInitImage(ximage);
  if (x_status == 0)
    {
      ximage=(XImage *) RelinquishMagickMemory(ximage);
      ThrowReaderException(CorruptImageError,"UnexpectedEndOfFile");
    }
  /*
    Read colormap.
  */
  authentic_colormap=MagickFalse;
  colors=(XColor *) NULL;
  if (header.ncolors != 0)
    {
      XWDColor
        color;

      colors=(XColor *) AcquireQuantumMemory((size_t) header.ncolors,
        sizeof(*colors));
      if (colors == (XColor *) NULL)
        {
          ximage=(XImage *) RelinquishMagickMemory(ximage);

ThrowReaderException(ResourceLimitError,"MemoryAllocationFailed");
        }
      for (i=0; i < (ssize_t) header.ncolors; i++)
      {
        count=ReadBlob(image,sz_XWDColor,(unsigned char *) &color);
        if (count != sz_XWDColor)
          {
            colors=(XColor *) RelinquishMagickMemory(colors);
            ximage=(XImage *) RelinquishMagickMemory(ximage);

ThrowReaderException(CorruptImageError,"UnexpectedEndOfFile");
          }
        colors[i].pixel=color.pixel;
        colors[i].red=color.red;
        colors[i].green=color.green;
        colors[i].blue=color.blue;
        colors[i].flags=(char) color.flags;
        if (color.flags != 0)
          authentic_colormap=MagickTrue;
      }
      /*
        Ensure the header byte-order is most-significant byte first.
```

```
      */
      lsb_first=1;
      if ((int) (*(char *) &lsb_first) != 0)
        for (i=0; i < (ssize_t) header.ncolors; i++)
        {
          MSBOrderLong((unsigned char *) &colors[i].pixel,
            sizeof(colors[i].pixel));
          MSBOrderShort((unsigned char *) &colors[i].red,3*
            sizeof(colors[i].red));
        }
    }
  /*
    Allocate the pixel buffer.
  */
  length=(size_t) ximage->bytes_per_line*ximage->height;
  if (CheckOverflowException(length,ximage->bytes_per_line,ximage-
>height))
    {
      if (header.ncolors != 0)
        colors=(XColor *) RelinquishMagickMemory(colors);
      ximage=(XImage *) RelinquishMagickMemory(ximage);
      ThrowReaderException(CorruptImageError,"ImproperImageHeader");
    }
  if (ximage->format != ZPixmap)
    {
      size_t
        extent;

      extent=length;
      length*=ximage->depth;
      if (CheckOverflowException(length,extent,ximage->depth))
        {
          if (header.ncolors != 0)
            colors=(XColor *) RelinquishMagickMemory(colors);
          ximage=(XImage *) RelinquishMagickMemory(ximage);
          ThrowReaderException(CorruptImageError,"ImproperImageHeader");
        }
    }
  ximage->data=(char *) AcquireQuantumMemory(length,sizeof(*ximage-
>data));
  if (ximage->data == (char *) NULL)
    {
      if (header.ncolors != 0)
        colors=(XColor *) RelinquishMagickMemory(colors);
      ximage=(XImage *) RelinquishMagickMemory(ximage);
      ThrowReaderException(ResourceLimitError,"MemoryAllocationFailed");
     }
  count=ReadBlob(image,length,(unsigned char *) ximage->data);
  if (count != (ssize_t) length)
    {
      if (header.ncolors != 0)
        colors=(XColor *) RelinquishMagickMemory(colors);
      ximage->data=DestroyString(ximage->data);
      ximage=(XImage *) RelinquishMagickMemory(ximage);
```

```
    ThrowReaderException(CorruptImageError,"UnableToReadImageData");
  }
/*
  Convert image to MIFF format.
*/
image->columns=(size_t) ximage->width;
image->rows=(size_t) ximage->height;
image->depth=8;
status=SetImageExtent(image,image->columns,image->rows);
if (status == MagickFalse)
  {
    if (header.ncolors != 0)
      colors=(XColor *) RelinquishMagickMemory(colors);
    ximage->data=DestroyString(ximage->data);
    ximage=(XImage *) RelinquishMagickMemory(ximage);
    InheritException(exception,&image->exception);
    return(DestroyImageList(image));
  }
if ((header.ncolors == 0U) || (ximage->red_mask != 0) ||
    (ximage->green_mask != 0) || (ximage->blue_mask != 0))
  image->storage_class=DirectClass;
else
  image->storage_class=PseudoClass;
image->colors=header.ncolors;
if (image_info->ping == MagickFalse)
  switch (image->storage_class)
  {
    case DirectClass:
    default:
    {
      register size_t
        color;

      size_t
        blue_mask,
        blue_shift,
        green_mask,
        green_shift,
        red_mask,
        red_shift;

      /*
        Determine shift and mask for red, green, and blue.
      */
      red_mask=ximage->red_mask;
      red_shift=0;
      while ((red_mask != 0) && ((red_mask & 0x01) == 0))
      {
        red_mask>>=1;
        red_shift++;
      }
      green_mask=ximage->green_mask;
      green_shift=0;
      while ((green_mask != 0) && ((green_mask & 0x01) == 0))
```

```
        {
          green_mask>>=1;
          green_shift++;
        }
        blue_mask=ximage->blue_mask;
        blue_shift=0;
        while ((blue_mask != 0) && ((blue_mask & 0x01) == 0))
        {
          blue_mask>>=1;
          blue_shift++;
        }
        /*
          Convert X image to DirectClass packets.
        */
        if ((image->colors != 0) && (authentic_colormap != MagickFalse))
          for (y=0; y < (ssize_t) image->rows; y++)
          {
            q=QueueAuthenticPixels(image,0,y,image->columns,1,exception);
            if (q == (PixelPacket *) NULL)
              break;
            for (x=0; x < (ssize_t) image->columns; x++)
            {
              pixel=XGetPixel(ximage,(int) x,(int) y);
              index=ConstrainColormapIndex(image,(ssize_t) (pixel >>
                red_shift) & red_mask);
              SetPixelRed(q,ScaleShortToQuantum(colors[(ssize_t)
index].red));
              index=ConstrainColormapIndex(image,(ssize_t) (pixel >>
                green_shift) & green_mask);
              SetPixelGreen(q,ScaleShortToQuantum(colors[(ssize_t)
                index].green));
              index=ConstrainColormapIndex(image,(ssize_t) (pixel >>
                blue_shift) & blue_mask);
              SetPixelBlue(q,ScaleShortToQuantum(colors[(ssize_t)
index].blue));
              q++;
            }
            if (SyncAuthenticPixels(image,exception) == MagickFalse)
              break;
            status=SetImageProgress(image,LoadImageTag,(MagickOffsetType)
y,
              image->rows);
            if (status == MagickFalse)
              break;
          }
        else
          for (y=0; y < (ssize_t) image->rows; y++)
          {
            q=QueueAuthenticPixels(image,0,y,image->columns,1,exception);
            if (q == (PixelPacket *) NULL)
              break;
            for (x=0; x < (ssize_t) image->columns; x++)
            {
              pixel=XGetPixel(ximage,(int) x,(int) y);
```

```
              color=(pixel >> red_shift) & red_mask;
              if (red_mask != 0)
                color=(color*65535UL)/red_mask;
              SetPixelRed(q,ScaleShortToQuantum((unsigned short) color));
              color=(pixel >> green_shift) & green_mask;
              if (green_mask != 0)
                color=(color*65535UL)/green_mask;
              SetPixelGreen(q,ScaleShortToQuantum((unsigned short)
color));
              color=(pixel >> blue_shift) & blue_mask;
              if (blue_mask != 0)
                color=(color*65535UL)/blue_mask;
              SetPixelBlue(q,ScaleShortToQuantum((unsigned short)
color));
              q++;
            }
            if (SyncAuthenticPixels(image,exception) == MagickFalse)
              break;
            status=SetImageProgress(image,LoadImageTag,(MagickOffsetType)
y,
              image->rows);
            if (status == MagickFalse)
              break;
          }
        break;
      }
      case PseudoClass:
      {
        /*
          Convert X image to PseudoClass packets.
        */
        if (AcquireImageColormap(image,image->colors) == MagickFalse)
          {
            if (header.ncolors != 0)
              colors=(XColor *) RelinquishMagickMemory(colors);
            ximage->data=DestroyString(ximage->data);
            ximage=(XImage *) RelinquishMagickMemory(ximage);

ThrowReaderException(ResourceLimitError,"MemoryAllocationFailed");
          }
        for (i=0; i < (ssize_t) image->colors; i++)
        {
          image->colormap[i].red=ScaleShortToQuantum(colors[i].red);
          image->colormap[i].green=ScaleShortToQuantum(colors[i].green);
          image->colormap[i].blue=ScaleShortToQuantum(colors[i].blue);
        }
        for (y=0; y < (ssize_t) image->rows; y++)
        {
          q=QueueAuthenticPixels(image,0,y,image->columns,1,exception);
          if (q == (PixelPacket *) NULL)
            break;
          indexes=GetAuthenticIndexQueue(image);
          for (x=0; x < (ssize_t) image->columns; x++)
          {
```

```
            index=ConstrainColormapIndex(image,(ssize_t)
XGetPixel(ximage,(int)
              x,(int) y));
            SetPixelIndex(indexes+x,index);
            SetPixelRGBO(q,image->colormap+(ssize_t) index);
            q++;
          }
          if (SyncAuthenticPixels(image,exception) == MagickFalse)
            break;
          status=SetImageProgress(image,LoadImageTag,(MagickOffsetType)
y,
            image->rows);
          if (status == MagickFalse)
            break;
        }
        break;
      }
    }
  /*
    Free image and colormap.
  */
  if (header.ncolors != 0)
    colors=(XColor *) RelinquishMagickMemory(colors);
  ximage->data=DestroyString(ximage->data);
  ximage=(XImage *) RelinquishMagickMemory(ximage);
  if (EOFBlob(image) != MagickFalse)
    ThrowFileException(exception,CorruptImageError,"UnexpectedEndOfFile",
      image->filename);
  (void) CloseBlob(image);
  return(GetFirstImageInList(image));
}
<sep>
static int trust_1oidany(X509_TRUST *trust, X509 *x, int flags)
{
    if (x->aux && (x->aux->trust || x->aux->reject))
        return obj_trust(trust->arg1, x, flags);
    /*
     * we don't have any trust settings: for compatibility we return
trusted
     * if it is self signed
     */
    return trust_compat(trust, x, flags);
}
<sep>
static int fts3IncrmergeLoad(
  Fts3Table *p,                   /* Fts3 table handle */
  sqlite3_int64 iAbsLevel,        /* Absolute level of input segments */
  int iIdx,                       /* Index of candidate output segment */
  const char *zKey,               /* First key to write */
  int nKey,                       /* Number of bytes in nKey */
  IncrmergeWriter *pWriter        /* Populate this object */
){
  int rc;                         /* Return code */
  sqlite3_stmt *pSelect = 0;      /* SELECT to read %_segdir entry */
```

```
  rc = fts3SqlStmt(p, SQL_SELECT_SEGDIR, &pSelect, 0);
  if( rc==SQLITE_OK ){
    sqlite3_int64 iStart = 0;     /* Value of %_segdir.start_block */
    sqlite3_int64 iLeafEnd = 0;   /* Value of %_segdir.leaves_end_block
*/
    sqlite3_int64 iEnd = 0;       /* Value of %_segdir.end_block */
    const char *aRoot = 0;        /* Pointer to %_segdir.root buffer */
    int nRoot = 0;                /* Size of aRoot[] in bytes */
    int rc2;                      /* Return code from sqlite3_reset() */
    int bAppendable = 0;          /* Set to true if segment is appendable
*/

    /* Read the %_segdir entry for index iIdx absolute level
(iAbsLevel+1) */
    sqlite3_bind_int64(pSelect, 1, iAbsLevel+1);
    sqlite3_bind_int(pSelect, 2, iIdx);
    if( sqlite3_step(pSelect)==SQLITE_ROW ){
      iStart = sqlite3_column_int64(pSelect, 1);
      iLeafEnd = sqlite3_column_int64(pSelect, 2);
      fts3ReadEndBlockField(pSelect, 3, &iEnd, &pWriter->nLeafData);
      if( pWriter->nLeafData<0 ){
        pWriter->nLeafData = pWriter->nLeafData * -1;
      }
      pWriter->bNoLeafData = (pWriter->nLeafData==0);
      nRoot = sqlite3_column_bytes(pSelect, 4);
      aRoot = sqlite3_column_blob(pSelect, 4);
    }else{
      return sqlite3_reset(pSelect);
    }

    /* Check for the zero-length marker in the %_segments table */
    rc = fts3IsAppendable(p, iEnd, &bAppendable);

    /* Check that zKey/nKey is larger than the largest key the candidate
*/
    if( rc==SQLITE_OK && bAppendable ){
      char *aLeaf = 0;
      int nLeaf = 0;

      rc = sqlite3Fts3ReadBlock(p, iLeafEnd, &aLeaf, &nLeaf, 0);
      if( rc==SQLITE_OK ){
        NodeReader reader;
        for(rc = nodeReaderInit(&reader, aLeaf, nLeaf);
            rc==SQLITE_OK && reader.aNode;
            rc = nodeReaderNext(&reader)
        ){
          assert( reader.aNode );
        }
        if( fts3TermCmp(zKey, nKey, reader.term.a, reader.term.n)<=0 ){
          bAppendable = 0;
        }
        nodeReaderRelease(&reader);
      }
```

```
      sqlite3_free(aLeaf);
    }

    if( rc==SQLITE_OK && bAppendable ){
      /* It is possible to append to this segment. Set up the
IncrmergeWriter
      ** object to do so.  */
      int i;
      int nHeight = (int)aRoot[0];
      NodeWriter *pNode;

      pWriter->nLeafEst = (int)((iEnd - iStart) +
1)/FTS_MAX_APPENDABLE_HEIGHT;
      pWriter->iStart = iStart;
      pWriter->iEnd = iEnd;
      pWriter->iAbsLevel = iAbsLevel;
      pWriter->iIdx = iIdx;

      for(i=nHeight+1; i<FTS_MAX_APPENDABLE_HEIGHT; i++){
        pWriter->aNodeWriter[i].iBlock = pWriter->iStart + i*pWriter-
>nLeafEst;
      }

      pNode = &pWriter->aNodeWriter[nHeight];
      pNode->iBlock = pWriter->iStart + pWriter->nLeafEst*nHeight;
      blobGrowBuffer(&pNode->block,
          MAX(nRoot, p->nNodeSize)+FTS3_NODE_PADDING, &rc
      );
      if( rc==SQLITE_OK ){
        memcpy(pNode->block.a, aRoot, nRoot);
        pNode->block.n = nRoot;
        memset(&pNode->block.a[nRoot], 0, FTS3_NODE_PADDING);
      }

      for(i=nHeight; i>=0 && rc==SQLITE_OK; i--){
        NodeReader reader;
        pNode = &pWriter->aNodeWriter[i];

        if( pNode->block.a){
          rc = nodeReaderInit(&reader, pNode->block.a, pNode->block.n);
          while( reader.aNode && rc==SQLITE_OK ) rc =
nodeReaderNext(&reader);
          blobGrowBuffer(&pNode->key, reader.term.n, &rc);
          if( rc==SQLITE_OK ){
            memcpy(pNode->key.a, reader.term.a, reader.term.n);
            pNode->key.n = reader.term.n;
            if( i>0 ){
              char *aBlock = 0;
              int nBlock = 0;
              pNode = &pWriter->aNodeWriter[i-1];
              pNode->iBlock = reader.iChild;
              rc = sqlite3Fts3ReadBlock(p, reader.iChild, &aBlock,
&nBlock, 0);
              blobGrowBuffer(&pNode->block,
```

```
                    MAX(nBlock, p->nNodeSize)+FTS3_NODE_PADDING, &rc
                );
                if( rc==SQLITE_OK ){
                  memcpy(pNode->block.a, aBlock, nBlock);
                  pNode->block.n = nBlock;
                  memset(&pNode->block.a[nBlock], 0, FTS3_NODE_PADDING);
                }
                sqlite3_free(aBlock);
              }
            }
          }
          nodeReaderRelease(&reader);
        }
      }

      rc2 = sqlite3_reset(pSelect);
      if( rc==SQLITE_OK ) rc = rc2;
    }

    return rc;
}
<sep>
cmdline_erase_chars(
      int c,
      int indent
#ifdef FEAT_SEARCH_EXTRA
      , incsearch_state_T *isp
#endif
      )
{
    int          i;
    int          j;

    if (c == K_KDEL)
      c = K_DEL;

    /*
     * Delete current character is the same as backspace on next
     * character, except at end of line.
     */
    if (c == K_DEL && ccline.cmdpos != ccline.cmdlen)
      ++ccline.cmdpos;
    if (has_mbyte && c == K_DEL)
      ccline.cmdpos += mb_off_next(ccline.cmdbuff,
            ccline.cmdbuff + ccline.cmdpos);
    if (ccline.cmdpos > 0)
    {
      char_u *p;

      j = ccline.cmdpos;
      p = ccline.cmdbuff + j;
      if (has_mbyte)
      {
          p = mb_prevptr(ccline.cmdbuff, p);
```

```
          if (c == Ctrl_W)
          {
            while (p > ccline.cmdbuff && vim_isspace(*p))
                p = mb_prevptr(ccline.cmdbuff, p);
            i = mb_get_class(p);
            while (p > ccline.cmdbuff && mb_get_class(p) == i)
                p = mb_prevptr(ccline.cmdbuff, p);
            if (mb_get_class(p) != i)
                p += (*mb_ptr2len)(p);
          }
      }
      else if (c == Ctrl_W)
      {
          while (p > ccline.cmdbuff && vim_isspace(p[-1]))
            --p;
          i = vim_iswordc(p[-1]);
          while (p > ccline.cmdbuff && !vim_isspace(p[-1])
                && vim_iswordc(p[-1]) == i)
            --p;
      }
      else
          --p;
      ccline.cmdpos = (int)(p - ccline.cmdbuff);
      ccline.cmdlen -= j - ccline.cmdpos;
      i = ccline.cmdpos;
      while (i < ccline.cmdlen)
          ccline.cmdbuff[i++] = ccline.cmdbuff[j++];

      // Truncate at the end, required for multi-byte chars.
      ccline.cmdbuff[ccline.cmdlen] = NUL;
#ifdef FEAT_SEARCH_EXTRA
      if (ccline.cmdlen == 0)
      {
          isp->search_start = isp->save_cursor;
          // save view settings, so that the screen
          // won't be restored at the wrong position
          isp->old_viewstate = isp->init_viewstate;
      }
#endif
      redrawcmd();
    }
    else if (ccline.cmdlen == 0 && c != Ctrl_W
          && ccline.cmdprompt == NULL && indent == 0)
    {
      // In ex and debug mode it doesn't make sense to return.
      if (exmode_active
#ifdef FEAT_EVAL
            || ccline.cmdfirstc == '>'
#endif
         )
          return CMDLINE_NOT_CHANGED;

      VIM_CLEAR(ccline.cmdbuff);   // no commandline to return
      if (!cmd_silent)
```

```
      {
#ifdef FEAT_RIGHTLEFT
          if (cmdmsg_rl)
            msg_col = Columns;
          else
#endif
            msg_col = 0;
          msg_putchar(' ');        // delete ':'
      }
#ifdef FEAT_SEARCH_EXTRA
      if (ccline.cmdlen == 0)
          isp->search_start = isp->save_cursor;
#endif
      redraw_cmdline = TRUE;
      return GOTO_NORMAL_MODE;
    }
    return CMDLINE_CHANGED;
}
<sep>
int ParseDsdiffHeaderConfig (FILE *infile, char *infilename, char
*fourcc, WavpackContext *wpc, WavpackConfig *config)
{
    int64_t infilesize, total_samples;
    DFFFileHeader dff_file_header;
    DFFChunkHeader dff_chunk_header;
    uint32_t bcount;

    infilesize = DoGetFileSize (infile);
    memcpy (&dff_file_header, fourcc, 4);

    if ((!DoReadFile (infile, ((char *) &dff_file_header) + 4, sizeof
(DFFFileHeader) - 4, &bcount) ||
        bcount != sizeof (DFFFileHeader) - 4) || strncmp
(dff_file_header.formType, "DSD ", 4)) {
            error_line ("%s is not a valid .DFF file!", infilename);
            return WAVPACK_SOFT_ERROR;
    }
    else if (!(config->qmode & QMODE_NO_STORE_WRAPPER) &&
        !WavpackAddWrapper (wpc, &dff_file_header, sizeof
(DFFFileHeader))) {
            error_line ("%s", WavpackGetErrorMessage (wpc));
            return WAVPACK_SOFT_ERROR;
    }

#if 1   // this might be a little too picky...
    WavpackBigEndianToNative (&dff_file_header, DFFFileHeaderFormat);

    if (infilesize && !(config->qmode & QMODE_IGNORE_LENGTH) &&
        dff_file_header.ckDataSize && dff_file_header.ckDataSize + 1 &&
dff_file_header.ckDataSize + 12 != infilesize) {
            error_line ("%s is not a valid .DFF file (by total size)!",
infilename);
            return WAVPACK_SOFT_ERROR;
    }
```

```
    if (debug_logging_mode)
        error_line ("file header indicated length = %lld",
dff_file_header.ckDataSize);

#endif

    // loop through all elements of the DSDIFF header
    // (until the data chuck) and copy them to the output file

    while (1) {
        if (!DoReadFile (infile, &dff_chunk_header, sizeof
(DFFChunkHeader), &bcount) ||
            bcount != sizeof (DFFChunkHeader)) {
                error_line ("%s is not a valid .DFF file!", infilename);
                return WAVPACK_SOFT_ERROR;
        }
        else if (!(config->qmode & QMODE_NO_STORE_WRAPPER) &&
            !WavpackAddWrapper (wpc, &dff_chunk_header, sizeof
(DFFChunkHeader))) {
                error_line ("%s", WavpackGetErrorMessage (wpc));
                return WAVPACK_SOFT_ERROR;
        }

        WavpackBigEndianToNative (&dff_chunk_header,
DFFChunkHeaderFormat);

        if (debug_logging_mode)
            error_line ("chunk header indicated length = %lld",
dff_chunk_header.ckDataSize);

        if (!strncmp (dff_chunk_header.ckID, "FVER", 4)) {
            uint32_t version;

            if (dff_chunk_header.ckDataSize != sizeof (version) ||
                !DoReadFile (infile, &version, sizeof (version), &bcount)
||
                bcount != sizeof (version)) {
                    error_line ("%s is not a valid .DFF file!",
infilename);
                    return WAVPACK_SOFT_ERROR;
            }
            else if (!(config->qmode & QMODE_NO_STORE_WRAPPER) &&
                !WavpackAddWrapper (wpc, &version, sizeof (version))) {
                    error_line ("%s", WavpackGetErrorMessage (wpc));
                    return WAVPACK_SOFT_ERROR;
            }

            WavpackBigEndianToNative (&version, "L");

            if (debug_logging_mode)
                error_line ("dsdiff file version = 0x%08x", version);
        }
        else if (!strncmp (dff_chunk_header.ckID, "PROP", 4)) {
```

```
            char *prop_chunk;

            if (dff_chunk_header.ckDataSize < 4 ||
dff_chunk_header.ckDataSize > 1024) {
                error_line ("%s is not a valid .DFF file!", infilename);
                return WAVPACK_SOFT_ERROR;
            }

            if (debug_logging_mode)
                error_line ("got PROP chunk of %d bytes total", (int)
dff_chunk_header.ckDataSize);

            prop_chunk = malloc ((size_t) dff_chunk_header.ckDataSize);

            if (!DoReadFile (infile, prop_chunk, (uint32_t)
dff_chunk_header.ckDataSize, &bcount) ||
                bcount != dff_chunk_header.ckDataSize) {
                    error_line ("%s is not a valid .DFF file!",
infilename);
                    free (prop_chunk);
                    return WAVPACK_SOFT_ERROR;
            }
            else if (!(config->qmode & QMODE_NO_STORE_WRAPPER) &&
                !WavpackAddWrapper (wpc, prop_chunk, (uint32_t)
dff_chunk_header.ckDataSize)) {
                    error_line ("%s", WavpackGetErrorMessage (wpc));
                    free (prop_chunk);
                    return WAVPACK_SOFT_ERROR;
            }

            if (!strncmp (prop_chunk, "SND ", 4)) {
                char *cptr = prop_chunk + 4, *eptr = prop_chunk +
dff_chunk_header.ckDataSize;
                uint16_t numChannels = 0, chansSpecified, chanMask = 0;
                uint32_t sampleRate = 0;

                while (eptr - cptr >= sizeof (dff_chunk_header)) {
                    memcpy (&dff_chunk_header, cptr, sizeof
(dff_chunk_header));
                    cptr += sizeof (dff_chunk_header);
                    WavpackBigEndianToNative (&dff_chunk_header,
DFFChunkHeaderFormat);

                    if (dff_chunk_header.ckDataSize > 0 &&
dff_chunk_header.ckDataSize <= eptr - cptr) {
                        if (!strncmp (dff_chunk_header.ckID, "FS  ", 4)
&& dff_chunk_header.ckDataSize == 4) {
                            memcpy (&sampleRate, cptr, sizeof
(sampleRate));
                            WavpackBigEndianToNative (&sampleRate, "L");
                            cptr += dff_chunk_header.ckDataSize;

                            if (debug_logging_mode)
```

```
                                error_line ("got sample rate of %u Hz",
sampleRate);
                        }
                        else if (!strncmp (dff_chunk_header.ckID, "CHNL",
4) && dff_chunk_header.ckDataSize >= 2) {
                            memcpy (&numChannels, cptr, sizeof
(numChannels));
                            WavpackBigEndianToNative (&numChannels, "S");
                            cptr += sizeof (numChannels);

                            chansSpecified =
(int)(dff_chunk_header.ckDataSize - sizeof (numChannels)) / 4;

                            if (numChannels < chansSpecified ||
numChannels < 1 || numChannels > 256) {
                                error_line ("%s is not a valid .DFF
file!", infilename);
                                free (prop_chunk);
                                return WAVPACK_SOFT_ERROR;
                            }

                            while (chansSpecified--) {
                                if (!strncmp (cptr, "SLFT", 4) ||
!strncmp (cptr, "MLFT", 4))
                                    chanMask |= 0x1;
                                else if (!strncmp (cptr, "SRGT", 4) ||
!strncmp (cptr, "MRGT", 4))
                                    chanMask |= 0x2;
                                else if (!strncmp (cptr, "LS  ", 4))
                                    chanMask |= 0x10;
                                else if (!strncmp (cptr, "RS  ", 4))
                                    chanMask |= 0x20;
                                else if (!strncmp (cptr, "C   ", 4))
                                    chanMask |= 0x4;
                                else if (!strncmp (cptr, "LFE ", 4))
                                    chanMask |= 0x8;
                                else
                                    if (debug_logging_mode)
                                        error_line ("undefined channel ID
%c%c%c%c", cptr [0], cptr [1], cptr [2], cptr [3]);

                                cptr += 4;
                            }

                            if (debug_logging_mode)
                                error_line ("%d channels, mask = 0x%08x",
numChannels, chanMask);
                        }
                        else if (!strncmp (dff_chunk_header.ckID, "CMPR",
4) && dff_chunk_header.ckDataSize >= 4) {
                            if (strncmp (cptr, "DSD ", 4)) {
                                error_line ("DSDIFF files must be
uncompressed, not \"%c%c%c%c\"!",
```

```
                                    cptr [0], cptr [1], cptr [2], cptr
[3]);
                                free (prop_chunk);
                                return WAVPACK_SOFT_ERROR;
                            }

                            cptr += dff_chunk_header.ckDataSize;
                        }
                        else {
                            if (debug_logging_mode)
                                error_line ("got PROP/SND chunk type
\"%c%c%c%c\" of %d bytes", dff_chunk_header.ckID [0],
                                    dff_chunk_header.ckID [1],
dff_chunk_header.ckID [2], dff_chunk_header.ckID [3],
dff_chunk_header.ckDataSize);

                            cptr += dff_chunk_header.ckDataSize;
                        }
                    }
                    else {
                        error_line ("%s is not a valid .DFF file!",
infilename);
                        free (prop_chunk);
                        return WAVPACK_SOFT_ERROR;
                    }
                }

                if (chanMask && (config->channel_mask || (config->qmode &
QMODE_CHANS_UNASSIGNED))) {
                    error_line ("this DSDIFF file already has channel
order information!");
                    free (prop_chunk);
                    return WAVPACK_SOFT_ERROR;
                }
                else if (chanMask)
                    config->channel_mask = chanMask;

                config->bits_per_sample = 8;
                config->bytes_per_sample = 1;
                config->num_channels = numChannels;
                config->sample_rate = sampleRate / 8;
                config->qmode |= QMODE_DSD_MSB_FIRST;
            }
            else if (debug_logging_mode)
                error_line ("got unknown PROP chunk type \"%c%c%c%c\" of
%d bytes",
                    prop_chunk [0], prop_chunk [1], prop_chunk [2],
prop_chunk [3], dff_chunk_header.ckDataSize);

            free (prop_chunk);
        }
        else if (!strncmp (dff_chunk_header.ckID, "DSD ", 4)) {

            if (!config->num_channels || !config->sample_rate) {
```

```
                error_line ("%s is not a valid .DFF file!", infilename);
                return WAVPACK_SOFT_ERROR;
            }

            total_samples = dff_chunk_header.ckDataSize / config-
>num_channels;
            break;
        }
        else {          // just copy unknown chunks to output file

            int bytes_to_copy = (int)(((dff_chunk_header.ckDataSize) + 1)
& ~(int64_t)1);
            char *buff;

            if (bytes_to_copy < 0 || bytes_to_copy > 4194304) {
                error_line ("%s is not a valid .DFF file!", infilename);
                return WAVPACK_SOFT_ERROR;
            }

            buff = malloc (bytes_to_copy);

            if (debug_logging_mode)
                error_line ("extra unknown chunk \"%c%c%c%c\" of %d
bytes",
                    dff_chunk_header.ckID [0], dff_chunk_header.ckID [1],
dff_chunk_header.ckID [2],
                    dff_chunk_header.ckID [3],
dff_chunk_header.ckDataSize);

            if (!DoReadFile (infile, buff, bytes_to_copy, &bcount) ||
                bcount != bytes_to_copy ||
                (!(config->qmode & QMODE_NO_STORE_WRAPPER) &&
                !WavpackAddWrapper (wpc, buff, bytes_to_copy))) {
                    error_line ("%s", WavpackGetErrorMessage (wpc));
                    free (buff);
                    return WAVPACK_SOFT_ERROR;
            }

            free (buff);
        }
    }

    if (debug_logging_mode)
        error_line ("setting configuration with %lld samples",
total_samples);

    if (!WavpackSetConfiguration64 (wpc, config, total_samples, NULL)) {
        error_line ("%s: %s", infilename, WavpackGetErrorMessage (wpc));
        return WAVPACK_SOFT_ERROR;
    }

    return WAVPACK_NO_ERROR;
}
<sep>
```

```
ImagingNew(const char* mode, int xsize, int ysize)
{
    int bytes;
    Imaging im;

    if (strlen(mode) == 1) {
        if (mode[0] == 'F' || mode[0] == 'I')
            bytes = 4;
        else
            bytes = 1;
    } else
        bytes = strlen(mode); /* close enough */

    if ((int64_t) xsize * (int64_t) ysize <= THRESHOLD / bytes) {
        im = ImagingNewBlock(mode, xsize, ysize);
        if (im)
            return im;
        /* assume memory error; try allocating in array mode instead */
        ImagingError_Clear();
    }

    return ImagingNewArray(mode, xsize, ysize);
}
<sep>
int socket_create(uint16_t port)
{
      int sfd = -1;
      int yes = 1;
#ifdef WIN32
      WSADATA wsa_data;
      if (!wsa_init) {
            if (WSAStartup(MAKEWORD(2,2), &wsa_data) != ERROR_SUCCESS) {
                  fprintf(stderr, "WSAStartup failed!\n");
                  ExitProcess(-1);
            }
            wsa_init = 1;
      }
#endif
      struct sockaddr_in saddr;

      if (0 > (sfd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP))) {
            perror("socket()");
            return -1;
      }

      if (setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR, (void*)&yes,
sizeof(int)) == -1) {
            perror("setsockopt()");
            socket_close(sfd);
            return -1;
      }

      memset((void *) &saddr, 0, sizeof(saddr));
      saddr.sin_family = AF_INET;
```

```
      saddr.sin_addr.s_addr = htonl(INADDR_ANY);
      saddr.sin_port = htons(port);

      if (0 > bind(sfd, (struct sockaddr *) &saddr, sizeof(saddr))) {
            perror("bind()");
            socket_close(sfd);
            return -1;
      }

      if (listen(sfd, 1) == -1) {
            perror("listen()");
            socket_close(sfd);
            return -1;
      }

      return sfd;
}
<sep>
find_help_tags(
    char_u  *arg,
    int          *num_matches,
    char_u  ***matches,
    int          keep_lang)
{
    char_u  *s, *d;
    int          i;
    // Specific tags that either have a specific replacement or won't go
    // through the generic rules.
    static char *(except_tbl[][2]) = {
      {"*",       "star"},
      {"g*",           "gstar"},
      {"[*",           "[star"},
      {"]*",           "]star"},
      {":*",           ":star"},
      {"/*",           "/star"},
      {"/\\*",    "/\\\\star"},
      {"\"*",          "quotestar"},
      {"**",           "starstar"},
      {"cpo-*",   "cpo-star"},
      {"/\\(\\)", "/\\\\(\\\\)"},
      {"/\\%(\\)",     "/\\\\%(\\\\)"},
      {"?",       "?"},
      {"??",           "??"},
      {":?",           ":?"},
      {"?<CR>",   "?<CR>"},
      {"g?",           "g?"},
      {"g?g?",    "g?g?"},
      {"g??",          "g??"},
      {"-?",           "-?"},
      {"q?",           "q?"},
      {"v_g?",    "v_g?"},
      {"/\\?",    "/\\\\?"},
      {"/\\z(\\)",     "/\\\\z(\\\\)"},
      {"\\=",          "\\\\="},
```

```
      {":s\\=",   ":s\\\\="},
      {"[count]", "\\[count]"},
      {"[quotex]",      "\\[quotex]"},
      {"[range]", "\\[range]"},
      {":[range]",      ":\\[range]"},
      {"[pattern]",     "\\[pattern]"},
      {"\\|",           "\\\\bar"},
      {"\\%$",    "/\\\\%\\$"},
      {"s/\\~",   "s/\\\\\\~"},
      {"s/\\U",   "s/\\\\U"},
      {"s/\\L",   "s/\\\\L"},
      {"s/\\1",   "s/\\\\1"},
      {"s/\\2",   "s/\\\\2"},
      {"s/\\3",   "s/\\\\3"},
      {"s/\\9",   "s/\\\\9"},
      {NULL, NULL}
    };
    static char *(expr_table[]) = {"!=?", "!~?", "<=?", "<?", "==?",
"=~?",
                           ">=?", ">?", "is?", "isnot?"};
    int flags;

    d = IObuff;            // assume IObuff is long enough!
    d[0] = NUL;

    if (STRNICMP(arg, "expr-", 5) == 0)
    {
      // When the string starting with "expr-" and containing '?' and
matches
      // the table, it is taken literally (but ~ is escaped).  Otherwise
'?'
      // is recognized as a wildcard.
      for (i = (int)ARRAY_LENGTH(expr_table); --i >= 0; )
          if (STRCMP(arg + 5, expr_table[i]) == 0)
          {
            int si = 0, di = 0;

            for (;;)
            {
                if (arg[si] == '~')
                  d[di++] = '\\';
                d[di++] = arg[si];
                if (arg[si] == NUL)
                  break;
                ++si;
            }
            break;
          }
    }
    else
    {
      // Recognize a few exceptions to the rule.  Some strings that
contain
```

```
      // '*'are changed to "star", otherwise '*' is recognized as a
wildcard.
      for (i = 0; except_tbl[i][0] != NULL; ++i)
          if (STRCMP(arg, except_tbl[i][0]) == 0)
          {
            STRCPY(d, except_tbl[i][1]);
            break;
          }
    }

    if (d[0] == NUL)   // no match in table
    {
      // Replace "\S" with "/\\S", etc.  Otherwise every tag is matched.
      // Also replace "\%^" and "\%(", they match every tag too.
      // Also "\zs", "\z1", etc.
      // Also "\@<", "\@=", "\@<=", etc.
      // And also "\_$" and "\_^".
      if (arg[0] == '\\'
            && ((arg[1] != NUL && arg[2] == NUL)
                || (vim_strchr((char_u *)"%_z@", arg[1]) != NULL
                                          && arg[2] != NUL)))
      {
          STRCPY(d, "/\\\\");
          STRCPY(d + 3, arg + 1);
          // Check for "/\\_$", should be "/\\_\$"
          if (d[3] == '_' && d[4] == '$')
            STRCPY(d + 4, "\\$");
      }
      else
      {
        // Replace:
        // "[:...:]" with "\[:...:]"
        // "[++...]" with "\[++...]"
        // "\{" with "\\{"           -- matching "} \}"
          if ((arg[0] == '[' && (arg[1] == ':'
                   || (arg[1] == '+' && arg[2] == '+')))
                || (arg[0] == '\\' && arg[1] == '{'))
            *d++ = '\\';

        // If tag starts with "('", skip the "(". Fixes CTRL-] on
('option'.
        if (*arg == '(' && arg[1] == '\'')
            arg++;
        for (s = arg; *s; ++s)
        {
          // Replace "|" with "bar" and '"' with "quote" to match the
name of
          // the tags for these commands.
          // Replace "*" with ".*" and "?" with "." to match command line
          // completion.
          // Insert a backslash before '~', '$' and '.' to avoid their
          // special meaning.
          if (d - IObuff > IOSIZE - 10) // getting too long!?
            break;
```

```
          switch (*s)
          {
            case '|':   STRCPY(d, "bar");
                        d += 3;
                        continue;
            case '"':   STRCPY(d, "quote");
                        d += 5;
                        continue;
            case '*':   *d++ = '.';
                        break;
            case '?':   *d++ = '.';
                        continue;
            case '$':
            case '.':
            case '~':   *d++ = '\\';
                        break;
          }

          // Replace "^x" by "CTRL-X". Don't do this for "^_" to make
          // ":help i_^_CTRL-D" work.
          // Insert '-' before and after "CTRL-X" when applicable.
          if (*s < ' ' || (*s == '^' && s[1] && (ASCII_ISALPHA(s[1])
                      || vim_strchr((char_u *)"?@[\\]^", s[1]) != NULL)))
          {
            if (d > IObuff && d[-1] != '_' && d[-1] != '\\')
                *d++ = '_';         // prepend a '_' to make x_CTRL-x
            STRCPY(d, "CTRL-");
            d += 5;
            if (*s < ' ')
            {
#ifdef EBCDIC
                *d++ = CtrlChar(*s);
#else
                *d++ = *s + '@';
#endif
                if (d[-1] == '\\')
                  *d++ = '\\';      // double a backslash
            }
            else
                *d++ = *++s;
            if (s[1] != NUL && s[1] != '_')
                *d++ = '_';         // append a '_'
            continue;
          }
          else if (*s == '^')           // "^" or "CTRL-^" or "^_"
            *d++ = '\\';

          // Insert a backslash before a backslash after a slash, for
search
          // pattern tags: "/\|" --> "/\\|".
          else if (s[0] == '\\' && s[1] != '\\'
                                    && *arg == '/' && s == arg + 1)
            *d++ = '\\';
```

```
          // "CTRL-\_" -> "CTRL-\\_" to avoid the special meaning of "\_"
in
          // "CTRL-\_CTRL-N"
          if (STRNICMP(s, "CTRL-\\_", 7) == 0)
          {
            STRCPY(d, "CTRL-\\\\");
            d += 7;
            s += 6;
          }

          *d++ = *s;

          // If tag contains "({" or "([", tag terminates at the "(".
          // This is for help on functions, e.g.: abs({expr}).
          if (*s == '(' && (s[1] == '{' || s[1] =='['))
            break;

          // If tag starts with ', toss everything after a second '.
Fixes
          // CTRL-] on 'option'. (would include the trailing '.').
          if (*s == '\'' && s > arg && *arg == '\'')
            break;
          // Also '{' and '}'.
          if (*s == '}' && s > arg && *arg == '{')
            break;
        }
        *d = NUL;

        if (*IObuff == '`')
        {
            if (d > IObuff + 2 && d[-1] == '`')
            {
              // remove the backticks from `command`
              mch_memmove(IObuff, IObuff + 1, STRLEN(IObuff));
              d[-2] = NUL;
            }
            else if (d > IObuff + 3 && d[-2] == '`' && d[-1] == ',')
            {
              // remove the backticks and comma from `command`,
              mch_memmove(IObuff, IObuff + 1, STRLEN(IObuff));
              d[-3] = NUL;
            }
            else if (d > IObuff + 4 && d[-3] == '`'
                                    && d[-2] == '\\' && d[-1] == '.')
            {
              // remove the backticks and dot from `command`\.
              mch_memmove(IObuff, IObuff + 1, STRLEN(IObuff));
              d[-4] = NUL;
            }
        }
      }
    }

    *matches = (char_u **)"";
```

```
    *num_matches = 0;
    flags = TAG_HELP | TAG_REGEXP | TAG_NAMES | TAG_VERBOSE |
TAG_NO_TAGFUNC;
    if (keep_lang)
      flags |= TAG_KEEP_LANG;
    if (find_tags(IObuff, num_matches, matches, flags, (int)MAXCOL, NULL)
== OK
          && *num_matches > 0)
    {
      // Sort the matches found on the heuristic number that is after the
      // tag name.
      qsort((void *)*matches, (size_t)*num_matches,
                                    sizeof(char_u *), help_compare);
      // Delete more than TAG_MANY to reduce the size of the listing.
      while (*num_matches > TAG_MANY)
          vim_free((*matches)[--*num_matches]);
    }
    return OK;
}
<sep>
QPDFWriter::calculateXrefStreamPadding(int xref_bytes)
{
    // This routine is called right after a linearization first pass
    // xref stream has been written without compression.  Calculate
    // the amount of padding that would be required in the worst case,
    // assuming the number of uncompressed bytes remains the same.
    // The worst case for zlib is that the output is larger than the
    // input by 6 bytes plus 5 bytes per 16K, and then we'll add 10
    // extra bytes for number length increases.

    return 16 + (5 * ((xref_bytes + 16383) / 16384));
}
<sep>
static coroutine_fn int nbd_negotiate(NBDClient *client, Error **errp)
{
    char buf[8 + 8 + 8 + 128];
    int ret;
    const uint16_t myflags = (NBD_FLAG_HAS_FLAGS | NBD_FLAG_SEND_TRIM |
                              NBD_FLAG_SEND_FLUSH | NBD_FLAG_SEND_FUA |
                              NBD_FLAG_SEND_WRITE_ZEROES);
    bool oldStyle;

    /* Old style negotiation header without options
        [ 0 ..   7]   passwd       ("NBDMAGIC")
        [ 8 ..  15]   magic        (NBD_CLIENT_MAGIC)
        [16 ..  23]   size
        [24 ..  25]   server flags (0)
        [26 ..  27]   export flags
        [28 .. 151]   reserved     (0)

       New style negotiation header with options
        [ 0 ..   7]   passwd       ("NBDMAGIC")
        [ 8 ..  15]   magic        (NBD_OPTS_MAGIC)
        [16 ..  17]   server flags (0)
```

```
         ....options sent, ending in NBD_OPT_EXPORT_NAME....
         */

        qio_channel_set_blocking(client->ioc, false, NULL);

        trace_nbd_negotiate_begin();
        memset(buf, 0, sizeof(buf));
        memcpy(buf, "NBDMAGIC", 8);

        oldStyle = client->exp != NULL && !client->tlscreds;
        if (oldStyle) {
            trace_nbd_negotiate_old_style(client->exp->size,
                                          client->exp->nbdflags | myflags);
            stq_be_p(buf + 8, NBD_CLIENT_MAGIC);
            stq_be_p(buf + 16, client->exp->size);
            stw_be_p(buf + 26, client->exp->nbdflags | myflags);

            if (nbd_write(client->ioc, buf, sizeof(buf), errp) < 0) {
                error_prepend(errp, "write failed: ");
                return -EINVAL;
            }
        } else {
            stq_be_p(buf + 8, NBD_OPTS_MAGIC);
            stw_be_p(buf + 16, NBD_FLAG_FIXED_NEWSTYLE | NBD_FLAG_NO_ZEROES);

            if (nbd_write(client->ioc, buf, 18, errp) < 0) {
                error_prepend(errp, "write failed: ");
                return -EINVAL;
            }
            ret = nbd_negotiate_options(client, myflags, errp);
            if (ret != 0) {
                if (ret < 0) {
                    error_prepend(errp, "option negotiation failed: ");
                }
                return ret;
            }
        }

        trace_nbd_negotiate_success();

        return 0;
    }
    <sep>
    static int usb_host_handle_control(USBHostDevice *s, USBPacket *p)
    {
        struct usbdevfs_urb *urb;
        AsyncURB *aurb;
        int ret, value, index;

        /*
         * Process certain standard device requests.
         * These are infrequent and are processed synchronously.
         */
        value = le16_to_cpu(s->ctrl.req.wValue);
```

```
    index = le16_to_cpu(s->ctrl.req.wIndex);

    dprintf("husb: ctrl type 0x%x req 0x%x val 0x%x index %u len %u\n",
        s->ctrl.req.bRequestType, s->ctrl.req.bRequest, value, index,
        s->ctrl.len);

    if (s->ctrl.req.bRequestType == 0) {
        switch (s->ctrl.req.bRequest) {
        case USB_REQ_SET_ADDRESS:
            return usb_host_set_address(s, value);

        case USB_REQ_SET_CONFIGURATION:
            return usb_host_set_config(s, value & 0xff);
        }
    }

    if (s->ctrl.req.bRequestType == 1 &&
                s->ctrl.req.bRequest == USB_REQ_SET_INTERFACE)
        return usb_host_set_interface(s, index, value);

    /* The rest are asynchronous */

    aurb = async_alloc();
    aurb->hdev   = s;
    aurb->packet = p;

    /*
     * Setup ctrl transfer.
     *
     * s->ctrl is layed out such that data buffer immediately follows
     * 'req' struct which is exactly what usbdevfs expects.
     */
    urb = &aurb->urb;

    urb->type     = USBDEVFS_URB_TYPE_CONTROL;
    urb->endpoint = p->devep;

    urb->buffer        = &s->ctrl.req;
    urb->buffer_length = 8 + s->ctrl.len;

    urb->usercontext = s;

    ret = ioctl(s->fd, USBDEVFS_SUBMITURB, urb);

    dprintf("husb: submit ctrl. len %u aurb %p\n", urb->buffer_length,
aurb);

    if (ret < 0) {
        dprintf("husb: submit failed. errno %d\n", errno);
        async_free(aurb);

        switch(errno) {
        case ETIMEDOUT:
            return USB_RET_NAK;
```

```
        case EPIPE:
        default:
            return USB_RET_STALL;
        }
    }

    usb_defer_packet(p, async_cancel, aurb);
    return USB_RET_ASYNC;
}
<sep>
PHP_MINIT_FUNCTION(xml)
{
      le_xml_parser =  zend_register_list_destructors_ex(xml_parser_dtor,
NULL, "xml", module_number);

      REGISTER_LONG_CONSTANT("XML_ERROR_NONE", XML_ERROR_NONE,
CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_NO_MEMORY", XML_ERROR_NO_MEMORY,
CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_SYNTAX", XML_ERROR_SYNTAX,
CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_NO_ELEMENTS",
XML_ERROR_NO_ELEMENTS, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_INVALID_TOKEN",
XML_ERROR_INVALID_TOKEN, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_UNCLOSED_TOKEN",
XML_ERROR_UNCLOSED_TOKEN, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_PARTIAL_CHAR",
XML_ERROR_PARTIAL_CHAR, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_TAG_MISMATCH",
XML_ERROR_TAG_MISMATCH, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_DUPLICATE_ATTRIBUTE",
XML_ERROR_DUPLICATE_ATTRIBUTE, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_JUNK_AFTER_DOC_ELEMENT",
XML_ERROR_JUNK_AFTER_DOC_ELEMENT, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_PARAM_ENTITY_REF",
XML_ERROR_PARAM_ENTITY_REF, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_UNDEFINED_ENTITY",
XML_ERROR_UNDEFINED_ENTITY, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_RECURSIVE_ENTITY_REF",
XML_ERROR_RECURSIVE_ENTITY_REF, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_ASYNC_ENTITY",
XML_ERROR_ASYNC_ENTITY, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_BAD_CHAR_REF",
XML_ERROR_BAD_CHAR_REF, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_BINARY_ENTITY_REF",
XML_ERROR_BINARY_ENTITY_REF, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF",
XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_MISPLACED_XML_PI",
XML_ERROR_MISPLACED_XML_PI, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_UNKNOWN_ENCODING",
XML_ERROR_UNKNOWN_ENCODING, CONST_CS|CONST_PERSISTENT);
```

```
      REGISTER_LONG_CONSTANT("XML_ERROR_INCORRECT_ENCODING",
XML_ERROR_INCORRECT_ENCODING, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_UNCLOSED_CDATA_SECTION",
XML_ERROR_UNCLOSED_CDATA_SECTION, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_ERROR_EXTERNAL_ENTITY_HANDLING",
XML_ERROR_EXTERNAL_ENTITY_HANDLING, CONST_CS|CONST_PERSISTENT);

      REGISTER_LONG_CONSTANT("XML_OPTION_CASE_FOLDING",
PHP_XML_OPTION_CASE_FOLDING, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_OPTION_TARGET_ENCODING",
PHP_XML_OPTION_TARGET_ENCODING, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_OPTION_SKIP_TAGSTART",
PHP_XML_OPTION_SKIP_TAGSTART, CONST_CS|CONST_PERSISTENT);
      REGISTER_LONG_CONSTANT("XML_OPTION_SKIP_WHITE",
PHP_XML_OPTION_SKIP_WHITE, CONST_CS|CONST_PERSISTENT);

      /* this object should not be pre-initialised at compile time,
         as the order of members may vary */

      php_xml_mem_hdlrs.malloc_fcn = php_xml_malloc_wrapper;
      php_xml_mem_hdlrs.realloc_fcn = php_xml_realloc_wrapper;
      php_xml_mem_hdlrs.free_fcn = php_xml_free_wrapper;

#ifdef LIBXML_EXPAT_COMPAT
      REGISTER_STRING_CONSTANT("XML_SAX_IMPL", "libxml",
CONST_CS|CONST_PERSISTENT);
#else
      REGISTER_STRING_CONSTANT("XML_SAX_IMPL", "expat",
CONST_CS|CONST_PERSISTENT);
#endif

      return SUCCESS;
}
<sep>
PHP_FUNCTION(exif_read_data)
{
      char *p_name, *p_sections_needed = NULL;
      size_t p_name_len, p_sections_needed_len = 0;
      zend_bool sub_arrays=0, read_thumbnail=0, read_all=0;

      int i, ret, sections_needed=0;
      image_info_type ImageInfo;
      char tmp[64], *sections_str, *s;

      if (zend_parse_parameters(ZEND_NUM_ARGS(), "p|sbb", &p_name,
&p_name_len, &p_sections_needed, &p_sections_needed_len, &sub_arrays,
&read_thumbnail) == FAILURE) {
            return;
      }

      memset(&ImageInfo, 0, sizeof(ImageInfo));

      if (p_sections_needed) {
            spprintf(&sections_str, 0, ",%s,", p_sections_needed);
```

```
            /* sections_str DOES start with , and SPACES are NOT allowed
in names */
            s = sections_str;
            while (*++s) {
                  if (*s == ' ') {
                        *s = ',';
                  }
            }

            for (i = 0; i < SECTION_COUNT; i++) {
                  snprintf(tmp, sizeof(tmp), ",%s,",
exif_get_sectionname(i));
                  if (strstr(sections_str, tmp)) {
                        sections_needed |= 1<<i;
                  }
            }
            EFREE_IF(sections_str);
            /* now see what we need */
#ifdef EXIF_DEBUG
            sections_str = exif_get_sectionlist(sections_needed);
            if (!sections_str) {
                  RETURN_FALSE;
            }
            exif_error_docref(NULL EXIFERR_CC, &ImageInfo, E_NOTICE,
"Sections needed: %s", sections_str[0] ? sections_str : "None");
            EFREE_IF(sections_str);
#endif
      }

      ret = exif_read_file(&ImageInfo, p_name, read_thumbnail, read_all);
      sections_str = exif_get_sectionlist(ImageInfo.sections_found);

#ifdef EXIF_DEBUG
      if (sections_str)
            exif_error_docref(NULL EXIFERR_CC, &ImageInfo, E_NOTICE,
"Sections found: %s", sections_str[0] ? sections_str : "None");
#endif

      ImageInfo.sections_found |= FOUND_COMPUTED|FOUND_FILE;/* do not
inform about in debug*/

      if (ret == FALSE || (sections_needed &&
!(sections_needed&ImageInfo.sections_found))) {
            /* array_init must be checked at last! otherwise the array
must be freed if a later test fails. */
            exif_discard_imageinfo(&ImageInfo);
            EFREE_IF(sections_str);
            RETURN_FALSE;
      }

      array_init(return_value);

#ifdef EXIF_DEBUG
```

```
      exif_error_docref(NULL EXIFERR_CC, &ImageInfo, E_NOTICE, "Generate
section FILE");
#endif

      /* now we can add our information */
      exif_iif_add_str(&ImageInfo, SECTION_FILE, "FileName",
ImageInfo.FileName);
      exif_iif_add_int(&ImageInfo, SECTION_FILE, "FileDateTime",
ImageInfo.FileDateTime);
      exif_iif_add_int(&ImageInfo, SECTION_FILE, "FileSize",
ImageInfo.FileSize);
      exif_iif_add_int(&ImageInfo, SECTION_FILE, "FileType",
ImageInfo.FileType);
      exif_iif_add_str(&ImageInfo, SECTION_FILE, "MimeType",
(char*)php_image_type_to_mime_type(ImageInfo.FileType));
      exif_iif_add_str(&ImageInfo, SECTION_FILE, "SectionsFound",
sections_str ? sections_str : "NONE");

#ifdef EXIF_DEBUG
      exif_error_docref(NULL EXIFERR_CC, &ImageInfo, E_NOTICE, "Generate
section COMPUTED");
#endif

      if (ImageInfo.Width>0 &&  ImageInfo.Height>0) {
            exif_iif_add_fmt(&ImageInfo, SECTION_COMPUTED, "html"   ,
"width=\"%d\" height=\"%d\"", ImageInfo.Width, ImageInfo.Height);
            exif_iif_add_int(&ImageInfo, SECTION_COMPUTED, "Height",
ImageInfo.Height);
            exif_iif_add_int(&ImageInfo, SECTION_COMPUTED, "Width",
ImageInfo.Width);
      }
      exif_iif_add_int(&ImageInfo, SECTION_COMPUTED, "IsColor",
ImageInfo.IsColor);
      if (ImageInfo.motorola_intel != -1) {
            exif_iif_add_int(&ImageInfo, SECTION_COMPUTED,
"ByteOrderMotorola", ImageInfo.motorola_intel);
      }
      if (ImageInfo.FocalLength) {
            exif_iif_add_fmt(&ImageInfo, SECTION_COMPUTED, "FocalLength",
"%4.1Fmm", ImageInfo.FocalLength);
            if(ImageInfo.CCDWidth) {
                  exif_iif_add_fmt(&ImageInfo, SECTION_COMPUTED,
"35mmFocalLength", "%dmm",
(int)(ImageInfo.FocalLength/ImageInfo.CCDWidth*35+0.5));
            }
      }
      if(ImageInfo.CCDWidth) {
            exif_iif_add_fmt(&ImageInfo, SECTION_COMPUTED, "CCDWidth",
"%dmm", (int)ImageInfo.CCDWidth);
      }
      if(ImageInfo.ExposureTime>0) {
            if(ImageInfo.ExposureTime <= 0.5) {
```

```
                exif_iif_add_fmt(&ImageInfo, SECTION_COMPUTED,
"ExposureTime", "%0.3F s (1/%d)", ImageInfo.ExposureTime, (int)(0.5 +
1/ImageInfo.ExposureTime));
           } else {
                exif_iif_add_fmt(&ImageInfo, SECTION_COMPUTED,
"ExposureTime", "%0.3F s", ImageInfo.ExposureTime);
           }
     }
     if(ImageInfo.ApertureFNumber) {
           exif_iif_add_fmt(&ImageInfo, SECTION_COMPUTED,
"ApertureFNumber", "f/%.1F", ImageInfo.ApertureFNumber);
     }
     if(ImageInfo.Distance) {
           if(ImageInfo.Distance<0) {
                exif_iif_add_str(&ImageInfo, SECTION_COMPUTED,
"FocusDistance", "Infinite");
           } else {
                exif_iif_add_fmt(&ImageInfo, SECTION_COMPUTED,
"FocusDistance", "%0.2Fm", ImageInfo.Distance);
           }
     }
     if (ImageInfo.UserComment) {
           exif_iif_add_buffer(&ImageInfo, SECTION_COMPUTED,
"UserComment", ImageInfo.UserCommentLength, ImageInfo.UserComment);
           if (ImageInfo.UserCommentEncoding &&
strlen(ImageInfo.UserCommentEncoding)) {
                exif_iif_add_str(&ImageInfo, SECTION_COMPUTED,
"UserCommentEncoding", ImageInfo.UserCommentEncoding);
           }
     }

     exif_iif_add_str(&ImageInfo, SECTION_COMPUTED, "Copyright",
ImageInfo.Copyright);
     exif_iif_add_str(&ImageInfo, SECTION_COMPUTED,
"Copyright.Photographer", ImageInfo.CopyrightPhotographer);
     exif_iif_add_str(&ImageInfo, SECTION_COMPUTED, "Copyright.Editor",
ImageInfo.CopyrightEditor);

     for (i=0; i<ImageInfo.xp_fields.count; i++) {
           exif_iif_add_str(&ImageInfo, SECTION_WINXP,
exif_get_tagname(ImageInfo.xp_fields.list[i].tag, NULL, 0,
exif_get_tag_table(SECTION_WINXP)), ImageInfo.xp_fields.list[i].value);
     }
     if (ImageInfo.Thumbnail.size) {
           if (read_thumbnail) {
                /* not exif_iif_add_str : this is a buffer */
                exif_iif_add_tag(&ImageInfo, SECTION_THUMBNAIL,
"THUMBNAIL", TAG_NONE, TAG_FMT_UNDEFINED, ImageInfo.Thumbnail.size,
ImageInfo.Thumbnail.data);
           }
           if (!ImageInfo.Thumbnail.width ||
!ImageInfo.Thumbnail.height) {
                /* try to evaluate if thumbnail data is present */
                exif_scan_thumbnail(&ImageInfo);
```

```
            }
            exif_iif_add_int(&ImageInfo, SECTION_COMPUTED,
"Thumbnail.FileType", ImageInfo.Thumbnail.filetype);
            exif_iif_add_str(&ImageInfo, SECTION_COMPUTED,
"Thumbnail.MimeType",
(char*)php_image_type_to_mime_type(ImageInfo.Thumbnail.filetype));
      }
      if (ImageInfo.Thumbnail.width && ImageInfo.Thumbnail.height) {
            exif_iif_add_int(&ImageInfo, SECTION_COMPUTED,
"Thumbnail.Height", ImageInfo.Thumbnail.height);
            exif_iif_add_int(&ImageInfo, SECTION_COMPUTED,
"Thumbnail.Width",  ImageInfo.Thumbnail.width);
      }
      EFREE_IF(sections_str);

#ifdef EXIF_DEBUG
      exif_error_docref(NULL EXIFERR_CC, &ImageInfo, E_NOTICE, "Adding
image infos");
#endif

      add_assoc_image_info(return_value, sub_arrays, &ImageInfo,
SECTION_FILE      );
      add_assoc_image_info(return_value, 1,          &ImageInfo,
SECTION_COMPUTED  );
      add_assoc_image_info(return_value, sub_arrays, &ImageInfo,
SECTION_ANY_TAG   );
      add_assoc_image_info(return_value, sub_arrays, &ImageInfo,
SECTION_IFD0      );
      add_assoc_image_info(return_value, 1,          &ImageInfo,
SECTION_THUMBNAIL );
      add_assoc_image_info(return_value, 1,          &ImageInfo,
SECTION_COMMENT   );
      add_assoc_image_info(return_value, sub_arrays, &ImageInfo,
SECTION_EXIF      );
      add_assoc_image_info(return_value, sub_arrays, &ImageInfo,
SECTION_GPS       );
      add_assoc_image_info(return_value, sub_arrays, &ImageInfo,
SECTION_INTEROP   );
      add_assoc_image_info(return_value, sub_arrays, &ImageInfo,
SECTION_FPIX      );
      add_assoc_image_info(return_value, sub_arrays, &ImageInfo,
SECTION_APP12     );
      add_assoc_image_info(return_value, sub_arrays, &ImageInfo,
SECTION_WINXP     );
      add_assoc_image_info(return_value, sub_arrays, &ImageInfo,
SECTION_MAKERNOTE );

#ifdef EXIF_DEBUG
      exif_error_docref(NULL EXIFERR_CC, &ImageInfo, E_NOTICE,
"Discarding info");
#endif

      exif_discard_imageinfo(&ImageInfo);
```

```
#ifdef EXIF_DEBUG
      php_error_docref1(NULL, p_name, E_NOTICE, "done");
#endif
}
<sep>
static int jpc_dec_process_siz(jpc_dec_t *dec, jpc_ms_t *ms)
{
      jpc_siz_t *siz = &ms->parms.siz;
      int compno;
      int tileno;
      jpc_dec_tile_t *tile;
      jpc_dec_tcomp_t *tcomp;
      int htileno;
      int vtileno;
      jpc_dec_cmpt_t *cmpt;

      dec->xstart = siz->xoff;
      dec->ystart = siz->yoff;
      dec->xend = siz->width;
      dec->yend = siz->height;
      dec->tilewidth = siz->tilewidth;
      dec->tileheight = siz->tileheight;
      dec->tilexoff = siz->tilexoff;
      dec->tileyoff = siz->tileyoff;
      dec->numcomps = siz->numcomps;
      if (!(dec->cp = jpc_dec_cp_create(dec->numcomps))) {
            return -1;
      }

      if (!(dec->cmpts = jas_alloc2(dec->numcomps,
sizeof(jpc_dec_cmpt_t)))) {
            return -1;
      }

      for (compno = 0, cmpt = dec->cmpts; compno < dec->numcomps;
++compno,
        ++cmpt) {
            cmpt->prec = siz->comps[compno].prec;
            cmpt->sgnd = siz->comps[compno].sgnd;
            cmpt->hstep = siz->comps[compno].hsamp;
            cmpt->vstep = siz->comps[compno].vsamp;
            cmpt->width = JPC_CEILDIV(dec->xend, cmpt->hstep) -
              JPC_CEILDIV(dec->xstart, cmpt->hstep);
            cmpt->height = JPC_CEILDIV(dec->yend, cmpt->vstep) -
              JPC_CEILDIV(dec->ystart, cmpt->vstep);
            cmpt->hsubstep = 0;
            cmpt->vsubstep = 0;
      }

      dec->image = 0;

      dec->numhtiles = JPC_CEILDIV(dec->xend - dec->tilexoff, dec-
>tilewidth);
```

```
      dec->numvtiles = JPC_CEILDIV(dec->yend - dec->tileyoff, dec-
>tileheight);
      dec->numtiles = dec->numhtiles * dec->numvtiles;
      if (!(dec->tiles = jas_alloc2(dec->numtiles,
sizeof(jpc_dec_tile_t)))) {
            return -1;
      }

      for (tileno = 0, tile = dec->tiles; tileno < dec->numtiles;
++tileno,
        ++tile) {
            htileno = tileno % dec->numhtiles;
            vtileno = tileno / dec->numhtiles;
            tile->realmode = 0;
            tile->state = JPC_TILE_INIT;
            tile->xstart = JAS_MAX(dec->tilexoff + htileno * dec-
>tilewidth,
              dec->xstart);
            tile->ystart = JAS_MAX(dec->tileyoff + vtileno * dec-
>tileheight,
              dec->ystart);
            tile->xend = JAS_MIN(dec->tilexoff + (htileno + 1) *
              dec->tilewidth, dec->xend);
            tile->yend = JAS_MIN(dec->tileyoff + (vtileno + 1) *
              dec->tileheight, dec->yend);
            tile->numparts = 0;
            tile->partno = 0;
            tile->pkthdrstream = 0;
            tile->pkthdrstreampos = 0;
            tile->pptstab = 0;
            tile->cp = 0;
            if (!(tile->tcomps = jas_alloc2(dec->numcomps,
              sizeof(jpc_dec_tcomp_t)))) {
                  return -1;
            }
            for (compno = 0, cmpt = dec->cmpts, tcomp = tile->tcomps;
              compno < dec->numcomps; ++compno, ++cmpt, ++tcomp) {
                  tcomp->rlvls = 0;
                  tcomp->numrlvls = 0;
                  tcomp->data = 0;
                  tcomp->xstart = JPC_CEILDIV(tile->xstart, cmpt->hstep);
                  tcomp->ystart = JPC_CEILDIV(tile->ystart, cmpt->vstep);
                  tcomp->xend = JPC_CEILDIV(tile->xend, cmpt->hstep);
                  tcomp->yend = JPC_CEILDIV(tile->yend, cmpt->vstep);
                  tcomp->tsfb = 0;
            }
      }

      dec->pkthdrstreams = 0;

      /* We should expect to encounter other main header marker segments
        or an SOT marker segment next. */
      dec->state = JPC_MH;
```

```
	return 0;
}
<sep>
static int rawv6_sendmsg(struct sock *sk, struct msghdr *msg, size_t len)
{
	struct ipv6_txoptions opt_space;
	DECLARE_SOCKADDR(struct sockaddr_in6 *, sin6, msg->msg_name);
	struct in6_addr *daddr, *final_p, final;
	struct inet_sock *inet = inet_sk(sk);
	struct ipv6_pinfo *np = inet6_sk(sk);
	struct raw6_sock *rp = raw6_sk(sk);
	struct ipv6_txoptions *opt = NULL;
	struct ip6_flowlabel *flowlabel = NULL;
	struct dst_entry *dst = NULL;
	struct raw6_frag_vec rfv;
	struct flowi6 fl6;
	int addr_len = msg->msg_namelen;
	int hlimit = -1;
	int tclass = -1;
	int dontfrag = -1;
	u16 proto;
	int err;

	/* Rough check on arithmetic overflow,
	   better check is made in ip6_append_data().
	 */
	if (len > INT_MAX)
		return -EMSGSIZE;

	/* Mirror BSD error message compatibility */
	if (msg->msg_flags & MSG_OOB)
		return -EOPNOTSUPP;

	/*
	 *	Get and verify the address.
	 */
	memset(&fl6, 0, sizeof(fl6));

	fl6.flowi6_mark = sk->sk_mark;

	if (sin6) {
		if (addr_len < SIN6_LEN_RFC2133)
			return -EINVAL;

		if (sin6->sin6_family && sin6->sin6_family != AF_INET6)
			return -EAFNOSUPPORT;

		/* port is the proto value [0..255] carried in nexthdr */
		proto = ntohs(sin6->sin6_port);

		if (!proto)
			proto = inet->inet_num;
		else if (proto != inet->inet_num)
			return -EINVAL;
```

```
      if (proto > 255)
            return -EINVAL;

      daddr = &sin6->sin6_addr;
      if (np->sndflow) {
            fl6.flowlabel = sin6->sin6_flowinfo&IPV6_FLOWINFO_MASK;
            if (fl6.flowlabel&IPV6_FLOWLABEL_MASK) {
                  flowlabel = fl6_sock_lookup(sk, fl6.flowlabel);
                  if (!flowlabel)
                        return -EINVAL;
            }
      }

      /*
       * Otherwise it will be difficult to maintain
       * sk->sk_dst_cache.
       */
      if (sk->sk_state == TCP_ESTABLISHED &&
          ipv6_addr_equal(daddr, &sk->sk_v6_daddr))
            daddr = &sk->sk_v6_daddr;

      if (addr_len >= sizeof(struct sockaddr_in6) &&
          sin6->sin6_scope_id &&
          __ipv6_addr_needs_scope_id(__ipv6_addr_type(daddr)))
            fl6.flowi6_oif = sin6->sin6_scope_id;
} else {
      if (sk->sk_state != TCP_ESTABLISHED)
            return -EDESTADDRREQ;

      proto = inet->inet_num;
      daddr = &sk->sk_v6_daddr;
      fl6.flowlabel = np->flow_label;
}

if (fl6.flowi6_oif == 0)
      fl6.flowi6_oif = sk->sk_bound_dev_if;

if (msg->msg_controllen) {
      opt = &opt_space;
      memset(opt, 0, sizeof(struct ipv6_txoptions));
      opt->tot_len = sizeof(struct ipv6_txoptions);

      err = ip6_datagram_send_ctl(sock_net(sk), sk, msg, &fl6, opt,
                                  &hlimit, &tclass, &dontfrag);
      if (err < 0) {
            fl6_sock_release(flowlabel);
            return err;
      }
      if ((fl6.flowlabel&IPV6_FLOWLABEL_MASK) && !flowlabel) {
            flowlabel = fl6_sock_lookup(sk, fl6.flowlabel);
            if (!flowlabel)
                  return -EINVAL;
      }
```

```
            if (!(opt->opt_nflen|opt->opt_flen))
                  opt = NULL;
      }
      if (!opt)
            opt = np->opt;
      if (flowlabel)
            opt = fl6_merge_options(&opt_space, flowlabel, opt);
      opt = ipv6_fixup_options(&opt_space, opt);

      fl6.flowi6_proto = proto;
      rfv.msg = msg;
      rfv.hlen = 0;
      err = rawv6_probe_proto_opt(&rfv, &fl6);
      if (err)
            goto out;

      if (!ipv6_addr_any(daddr))
            fl6.daddr = *daddr;
      else
            fl6.daddr.s6_addr[15] = 0x1; /* :: means loopback (BSD'ism)
*/
      if (ipv6_addr_any(&fl6.saddr) && !ipv6_addr_any(&np->saddr))
            fl6.saddr = np->saddr;

      final_p = fl6_update_dst(&fl6, opt, &final);

      if (!fl6.flowi6_oif && ipv6_addr_is_multicast(&fl6.daddr))
            fl6.flowi6_oif = np->mcast_oif;
      else if (!fl6.flowi6_oif)
            fl6.flowi6_oif = np->ucast_oif;
      security_sk_classify_flow(sk, flowi6_to_flowi(&fl6));

      if (inet->hdrincl)
            fl6.flowi6_flags |= FLOWI_FLAG_KNOWN_NH;

      dst = ip6_dst_lookup_flow(sk, &fl6, final_p);
      if (IS_ERR(dst)) {
            err = PTR_ERR(dst);
            goto out;
      }
      if (hlimit < 0)
            hlimit = ip6_sk_dst_hoplimit(np, &fl6, dst);

      if (tclass < 0)
            tclass = np->tclass;

      if (dontfrag < 0)
            dontfrag = np->dontfrag;

      if (msg->msg_flags&MSG_CONFIRM)
            goto do_confirm;

back_from_confirm:
      if (inet->hdrincl)
```

```
            err = rawv6_send_hdrinc(sk, msg, len, &fl6, &dst, msg-
>msg_flags);
      else {
            lock_sock(sk);
            err = ip6_append_data(sk, raw6_getfrag, &rfv,
                  len, 0, hlimit, tclass, opt, &fl6, (struct rt6_info
*)dst,
                  msg->msg_flags, dontfrag);

            if (err)
                  ip6_flush_pending_frames(sk);
            else if (!(msg->msg_flags & MSG_MORE))
                  err = rawv6_push_pending_frames(sk, &fl6, rp);
            release_sock(sk);
      }
done:
      dst_release(dst);
out:
      fl6_sock_release(flowlabel);
      return err < 0 ? err : len;
do_confirm:
      dst_confirm(dst);
      if (!(msg->msg_flags & MSG_PROBE) || len)
            goto back_from_confirm;
      err = 0;
      goto done;
}
<sep>
  void Compute(OpKernelContext* const context) override {
    core::RefCountPtr<BoostedTreesEnsembleResource> resource;
    // Get the resource.
    OP_REQUIRES_OK(context, LookupResource(context,
HandleFromInput(context, 0),
                                           &resource));

    // Get the inputs.
    OpInputList bucketized_features_list;
    OP_REQUIRES_OK(context, context->input_list("bucketized_features",

&bucketized_features_list));
    std::vector<tensorflow::TTypes<int32>::ConstMatrix>
bucketized_features;
    bucketized_features.reserve(bucketized_features_list.size());
    ConvertVectorsToMatrices(bucketized_features_list,
bucketized_features);
    const int batch_size = bucketized_features[0].dimension(0);

    // We need to get the feature ids used for splitting and the logits
after
    // each split. We will use these to calculate the changes in the
prediction
    // (contributions) for an arbitrary activation function (done in
Python) and
```

```
    // attribute them to the associated feature ids. We will store these
in
    // a proto below.
    Tensor* output_debug_info_t = nullptr;
    OP_REQUIRES_OK(
        context, context-
>allocate_output("examples_debug_outputs_serialized",
                                            {batch_size},
&output_debug_info_t));
    // Will contain serialized protos, per example.
    auto output_debug_info = output_debug_info_t->flat<tstring>();
    const int32 last_tree = resource->num_trees() - 1;

    // For each given example, traverse through all trees keeping track
of the
    // features used to split and the associated logits at each point
along the
    // path. Note: feature_ids has one less value than logits_path
because the
    // first value of each logit path will be the bias.
    auto do_work = [&resource, &bucketized_features, &output_debug_info,
                    last_tree](int32 start, int32 end) {
      for (int32 i = start; i < end; ++i) {
        // Proto to store debug outputs, per example.
        boosted_trees::DebugOutput example_debug_info;
        // Initial bias prediction. E.g., prediction based off training
mean.
        const auto& tree_logits = resource->node_value(0, 0);
        DCHECK_EQ(tree_logits.size(), 1);
        float tree_logit = resource->GetTreeWeight(0) * tree_logits[0];
        example_debug_info.add_logits_path(tree_logit);
        int32 node_id = 0;
        int32 tree_id = 0;
        int32 feature_id;
        float past_trees_logit = 0;  // Sum of leaf logits from prior
trees.
        // Go through each tree and populate proto.
        while (tree_id <= last_tree) {
          if (resource->is_leaf(tree_id, node_id)) {  // Move onto other
trees.
            // Accumulate tree_logits only if the leaf is non-root, but
do so
            // for bias tree.
            if (tree_id == 0 || node_id > 0) {
              past_trees_logit += tree_logit;
            }
            ++tree_id;
            node_id = 0;
          } else {  // Add to proto.
            // Feature id used to split.
            feature_id = resource->feature_id(tree_id, node_id);
            example_debug_info.add_feature_ids(feature_id);
            // Get logit after split.
            node_id =
```

```
                resource->next_node(tree_id, node_id, i,
bucketized_features);
            const auto& tree_logits = resource->node_value(tree_id,
node_id);
            DCHECK_EQ(tree_logits.size(), 1);
            tree_logit = resource->GetTreeWeight(tree_id) *
tree_logits[0];
            // Output logit incorporates sum of leaf logits from prior
trees.
            example_debug_info.add_logits_path(tree_logit +
past_trees_logit);
          }
        }
        // Set output as serialized proto containing debug info.
        string serialized = example_debug_info.SerializeAsString();
        output_debug_info(i) = serialized;
      }
    };

    // 10 is the magic number. The actual number might depend on (the
number of
    // layers in the trees) and (cpu cycles spent on each layer), but
this
    // value would work for many cases. May be tuned later.
    const int64 cost = (last_tree + 1) * 10;
    thread::ThreadPool* const worker_threads =
        context->device()->tensorflow_cpu_worker_threads()->workers;
    Shard(worker_threads->NumThreads(), worker_threads, batch_size,
          /*cost_per_unit=*/cost, do_work);
  }
<sep>
ssize_t qemu_sendv_packet_async(NetClientState *sender,
                                const struct iovec *iov, int iovcnt,
                                NetPacketSent *sent_cb)
{
    NetQueue *queue;
    int ret;

    if (sender->link_down || !sender->peer) {
        return iov_size(iov, iovcnt);
    }

    /* Let filters handle the packet first */
    ret = filter_receive_iov(sender, NET_FILTER_DIRECTION_TX, sender,
                             QEMU_NET_PACKET_FLAG_NONE, iov, iovcnt,
sent_cb);
    if (ret) {
        return ret;
    }

    ret = filter_receive_iov(sender->peer, NET_FILTER_DIRECTION_RX,
sender,
                             QEMU_NET_PACKET_FLAG_NONE, iov, iovcnt,
sent_cb);
```

```
    if (ret) {
        return ret;
    }

    queue = sender->peer->incoming_queue;

    return qemu_net_queue_send_iov(queue, sender,
                                   QEMU_NET_PACKET_FLAG_NONE,
                                   iov, iovcnt, sent_cb);
}
<sep>
int vmw_gb_surface_define_ioctl(struct drm_device *dev, void *data,
                        struct drm_file *file_priv)
{
      struct vmw_private *dev_priv = vmw_priv(dev);
      struct vmw_user_surface *user_srf;
      struct vmw_surface *srf;
      struct vmw_resource *res;
      struct vmw_resource *tmp;
      union drm_vmw_gb_surface_create_arg *arg =
          (union drm_vmw_gb_surface_create_arg *)data;
      struct drm_vmw_gb_surface_create_req *req = &arg->req;
      struct drm_vmw_gb_surface_create_rep *rep = &arg->rep;
      struct ttm_object_file *tfile = vmw_fpriv(file_priv)->tfile;
      int ret;
      uint32_t size;
      uint32_t backup_handle;

      if (req->multisample_count != 0)
            return -EINVAL;

      if (req->mip_levels > DRM_VMW_MAX_MIP_LEVELS)
            return -EINVAL;

      if (unlikely(vmw_user_surface_size == 0))
            vmw_user_surface_size = ttm_round_pot(sizeof(*user_srf)) +
                  128;

      size = vmw_user_surface_size + 128;

      /* Define a surface based on the parameters. */
      ret = vmw_surface_gb_priv_define(dev,
                  size,
                  req->svga3d_flags,
                  req->format,
                  req->drm_surface_flags & drm_vmw_surface_flag_scanout,
                  req->mip_levels,
                  req->multisample_count,
                  req->array_size,
                  req->base_size,
                  &srf);
      if (unlikely(ret != 0))
            return ret;
```

```
	user_srf = container_of(srf, struct vmw_user_surface, srf);
	if (drm_is_primary_client(file_priv))
		user_srf->master = drm_master_get(file_priv->master);

	ret = ttm_read_lock(&dev_priv->reservation_sem, true);
	if (unlikely(ret != 0))
		return ret;

	res = &user_srf->srf.res;


	if (req->buffer_handle != SVGA3D_INVALID_ID) {
		ret = vmw_user_dmabuf_lookup(tfile, req->buffer_handle,
		                             &res->backup,
		                             &user_srf->backup_base);
		if (ret == 0 && res->backup->base.num_pages * PAGE_SIZE <
		    res->backup_size) {
		  DRM_ERROR("Surface backup buffer is too small.\n");
		  vmw_dmabuf_unreference(&res->backup);
		  ret = -EINVAL;
		  goto out_unlock;
		}
	} else if (req->drm_surface_flags &
drm_vmw_surface_flag_create_buffer)
		ret = vmw_user_dmabuf_alloc(dev_priv, tfile,
		                            res->backup_size,
		                            req->drm_surface_flags &
		                            drm_vmw_surface_flag_shareable,
		                            &backup_handle,
		                            &res->backup,
		                            &user_srf->backup_base);

	if (unlikely(ret != 0)) {
		vmw_resource_unreference(&res);
		goto out_unlock;
	}

	tmp = vmw_resource_reference(res);
	ret = ttm_prime_object_init(tfile, res->backup_size, &user_srf-
>prime,
	                            req->drm_surface_flags &
	                            drm_vmw_surface_flag_shareable,
	                            VMW_RES_SURFACE,
	                            &vmw_user_surface_base_release, NULL);

	if (unlikely(ret != 0)) {
		vmw_resource_unreference(&tmp);
		vmw_resource_unreference(&res);
		goto out_unlock;
	}

	rep->handle      = user_srf->prime.base.hash.key;
	rep->backup_size = res->backup_size;
	if (res->backup) {
```

```
            rep->buffer_map_handle =
                  drm_vma_node_offset_addr(&res->backup->base.vma_node);
            rep->buffer_size = res->backup->base.num_pages * PAGE_SIZE;
            rep->buffer_handle = backup_handle;
      } else {
            rep->buffer_map_handle = 0;
            rep->buffer_size = 0;
            rep->buffer_handle = SVGA3D_INVALID_ID;
      }

      vmw_resource_unreference(&res);

out_unlock:
      ttm_read_unlock(&dev_priv->reservation_sem);
      return ret;
}
<sep>
static int crypto_report_one(struct crypto_alg *alg,
                     struct crypto_user_alg *ualg, struct sk_buff *skb)
{
      strlcpy(ualg->cru_name, alg->cra_name, sizeof(ualg->cru_name));
      strlcpy(ualg->cru_driver_name, alg->cra_driver_name,
            sizeof(ualg->cru_driver_name));
      strlcpy(ualg->cru_module_name, module_name(alg->cra_module),
            sizeof(ualg->cru_module_name));

      ualg->cru_type = 0;
      ualg->cru_mask = 0;
      ualg->cru_flags = alg->cra_flags;
      ualg->cru_refcnt = refcount_read(&alg->cra_refcnt);

      if (nla_put_u32(skb, CRYPTOCFGA_PRIORITY_VAL, alg->cra_priority))
            goto nla_put_failure;
      if (alg->cra_flags & CRYPTO_ALG_LARVAL) {
            struct crypto_report_larval rl;

            strlcpy(rl.type, "larval", sizeof(rl.type));
            if (nla_put(skb, CRYPTOCFGA_REPORT_LARVAL,
                        sizeof(struct crypto_report_larval), &rl))
                  goto nla_put_failure;
            goto out;
      }

      if (alg->cra_type && alg->cra_type->report) {
            if (alg->cra_type->report(skb, alg))
                  goto nla_put_failure;

            goto out;
      }

      switch (alg->cra_flags & (CRYPTO_ALG_TYPE_MASK |
CRYPTO_ALG_LARVAL)) {
      case CRYPTO_ALG_TYPE_CIPHER:
            if (crypto_report_cipher(skb, alg))
```

```
                goto nla_put_failure;

            break;
      case CRYPTO_ALG_TYPE_COMPRESS:
            if (crypto_report_comp(skb, alg))
                  goto nla_put_failure;

            break;
      case CRYPTO_ALG_TYPE_ACOMPRESS:
            if (crypto_report_acomp(skb, alg))
                  goto nla_put_failure;

            break;
      case CRYPTO_ALG_TYPE_AKCIPHER:
            if (crypto_report_akcipher(skb, alg))
                  goto nla_put_failure;

            break;
      case CRYPTO_ALG_TYPE_KPP:
            if (crypto_report_kpp(skb, alg))
                  goto nla_put_failure;
            break;
      }

out:
      return 0;

nla_put_failure:
      return -EMSGSIZE;
}
<sep>
static inline void aio_poll_complete(struct aio_kiocb *iocb, __poll_t
mask)
{
      struct file *file = iocb->poll.file;

      aio_complete(iocb, mangle_poll(mask), 0);
      fput(file);
}
<sep>
long vt_compat_ioctl(struct tty_struct *tty,
           unsigned int cmd, unsigned long arg)
{
      struct vc_data *vc = tty->driver_data;
      struct console_font_op op;   /* used in multiple places here */
      void __user *up = compat_ptr(arg);
      int perm;

      /*
       * To have permissions to do most of the vt ioctls, we either have
       * to be the owner of the tty, or have CAP_SYS_TTY_CONFIG.
       */
      perm = 0;
      if (current->signal->tty == tty || capable(CAP_SYS_TTY_CONFIG))
```

```
            perm = 1;

      switch (cmd) {
      /*
       * these need special handlers for incompatible data structures
       */
      case PIO_FONTX:
      case GIO_FONTX:
            return compat_fontx_ioctl(cmd, up, perm, &op);

      case KDFONTOP:
            return compat_kdfontop_ioctl(up, perm, &op, vc);

      case PIO_UNIMAP:
      case GIO_UNIMAP:
            return compat_unimap_ioctl(cmd, up, perm, vc);

      /*
       * all these treat 'arg' as an integer
       */
      case KIOCSOUND:
      case KDMKTONE:
#ifdef CONFIG_X86
      case KDADDIO:
      case KDDELIO:
#endif
      case KDSETMODE:
      case KDMAPDISP:
      case KDUNMAPDISP:
      case KDSKBMODE:
      case KDSKBMETA:
      case KDSKBLED:
      case KDSETLED:
      case KDSIGACCEPT:
      case VT_ACTIVATE:
      case VT_WAITACTIVE:
      case VT_RELDISP:
      case VT_DISALLOCATE:
      case VT_RESIZE:
      case VT_RESIZEX:
            return vt_ioctl(tty, cmd, arg);

      /*
       * the rest has a compatible data structure behind arg,
       * but we have to convert it to a proper 64 bit pointer.
       */
      default:
            return vt_ioctl(tty, cmd, (unsigned long)up);
      }
}
<sep>
static inline int zpff_init(struct hid_device *hid)
{
      return 0;
```

```
}
<sep>
parse_tsquery(char *buf,
                  PushFunction pushval,
                  Datum opaque,
                  bool isplain)
{
      struct TSQueryParserStateData state;
      int               i;
      TSQuery           query;
      int               commonlen;
      QueryItem  *ptr;
      ListCell   *cell;

      /* init state */
      state.buffer = buf;
      state.buf = buf;
      state.state = (isplain) ? WAITSINGLEOPERAND : WAITFIRSTOPERAND;
      state.count = 0;
      state.polstr = NIL;

      /* init value parser's state */
      state.valstate = init_tsvector_parser(state.buffer, true, true);

      /* init list of operand */
      state.sumlen = 0;
      state.lenop = 64;
      state.curop = state.op = (char *) palloc(state.lenop);
      *(state.curop) = '\0';

      /* parse query & make polish notation (postfix, but in reverse
order) */
      makepol(&state, pushval, opaque);

      close_tsvector_parser(state.valstate);

      if (list_length(state.polstr) == 0)
      {
            ereport(NOTICE,
                        (errmsg("text-search query doesn't contain
lexemes: \"%s\"",
                                    state.buffer)));
            query = (TSQuery) palloc(HDRSIZETQ);
            SET_VARSIZE(query, HDRSIZETQ);
            query->size = 0;
            return query;
      }

      /* Pack the QueryItems in the final TSQuery struct to return to
caller */
      commonlen = COMPUTESIZE(list_length(state.polstr), state.sumlen);
      query = (TSQuery) palloc0(commonlen);
      SET_VARSIZE(query, commonlen);
      query->size = list_length(state.polstr);
```

```
      ptr = GETQUERY(query);

      /* Copy QueryItems to TSQuery */
      i = 0;
      foreach(cell, state.polstr)
      {
            QueryItem  *item = (QueryItem *) lfirst(cell);

            switch (item->type)
            {
                  case QI_VAL:
                        memcpy(&ptr[i], item, sizeof(QueryOperand));
                        break;
                  case QI_VALSTOP:
                        ptr[i].type = QI_VALSTOP;
                        break;
                  case QI_OPR:
                        memcpy(&ptr[i], item, sizeof(QueryOperator));
                        break;
                  default:
                        elog(ERROR, "unrecognized QueryItem type: %d",
item->type);
            }
            i++;
      }

      /* Copy all the operand strings to TSQuery */
      memcpy((void *) GETOPERAND(query), (void *) state.op,
state.sumlen);
      pfree(state.op);

      /* Set left operand pointers for every operator. */
      findoprnd(ptr, query->size);

      return query;
}
<sep>
_gcry_ecc_gost_sign (gcry_mpi_t input, ECC_secret_key *skey,
                     gcry_mpi_t r, gcry_mpi_t s)
{
  gpg_err_code_t rc = 0;
  gcry_mpi_t k, dr, sum, ke, x, e;
  mpi_point_struct I;
  gcry_mpi_t hash;
  const void *abuf;
  unsigned int abits, qbits;
  mpi_ec_t ctx;

  if (DBG_CIPHER)
    log_mpidump ("gost sign hash  ", input );

  qbits = mpi_get_nbits (skey->E.n);

  /* Convert the INPUT into an MPI if needed.  */
```

```
  if (mpi_is_opaque (input))
    {
      abuf = mpi_get_opaque (input, &abits);
      rc = _gcry_mpi_scan (&hash, GCRYMPI_FMT_USG, abuf, (abits+7)/8,
NULL);
      if (rc)
        return rc;
      if (abits > qbits)
        mpi_rshift (hash, hash, abits - qbits);
    }
  else
    hash = input;


  k = NULL;
  dr = mpi_alloc (0);
  sum = mpi_alloc (0);
  ke = mpi_alloc (0);
  e = mpi_alloc (0);
  x = mpi_alloc (0);
  point_init (&I);

  ctx = _gcry_mpi_ec_p_internal_new (skey->E.model, skey->E.dialect, 0,
                                     skey->E.p, skey->E.a, skey->E.b);

  mpi_mod (e, input, skey->E.n); /* e = hash mod n */

  if (!mpi_cmp_ui (e, 0))
    mpi_set_ui (e, 1);

  /* Two loops to avoid R or S are zero.  This is more of a joke than
     a real demand because the probability of them being zero is less
     than any hardware failure.  Some specs however require it.  */
  do
    {
      do
        {
          mpi_free (k);
          k = _gcry_dsa_gen_k (skey->E.n, GCRY_STRONG_RANDOM);

          _gcry_mpi_ec_mul_point (&I, k, &skey->E.G, ctx);
          if (_gcry_mpi_ec_get_affine (x, NULL, &I, ctx))
            {
              if (DBG_CIPHER)
                log_debug ("ecc sign: Failed to get affine
coordinates\n");
              rc = GPG_ERR_BAD_SIGNATURE;
              goto leave;
            }
          mpi_mod (r, x, skey->E.n);  /* r = x mod n */
        }
      while (!mpi_cmp_ui (r, 0));
      mpi_mulm (dr, skey->d, r, skey->E.n); /* dr = d*r mod n  */
      mpi_mulm (ke, k, e, skey->E.n); /* ke = k*e mod n */
```

```
      mpi_addm (s, ke, dr, skey->E.n); /* sum = (k*e+ d*r) mod n  */
    }
  while (!mpi_cmp_ui (s, 0));

  if (DBG_CIPHER)
    {
      log_mpidump ("gost sign result r ", r);
      log_mpidump ("gost sign result s ", s);
    }

 leave:
  _gcry_mpi_ec_free (ctx);
  point_free (&I);
  mpi_free (x);
  mpi_free (e);
  mpi_free (ke);
  mpi_free (sum);
  mpi_free (dr);
  mpi_free (k);

  if (hash != input)
    mpi_free (hash);

  return rc;
}
<sep>
svcauth_gss_accept_sec_context(struct svc_req *rqst,
                        struct rpc_gss_init_res *gr)
{
      struct svc_rpc_gss_data      *gd;
      struct rpc_gss_cred    *gc;
      gss_buffer_desc         recv_tok, seqbuf;
      gss_OID                 mech;
      OM_uint32         maj_stat = 0, min_stat = 0, ret_flags, seq;

      log_debug("in svcauth_gss_accept_context()");

      gd = SVCAUTH_PRIVATE(rqst->rq_xprt->xp_auth);
      gc = (struct rpc_gss_cred *)rqst->rq_clntcred;
      memset(gr, 0, sizeof(*gr));

      /* Deserialize arguments. */
      memset(&recv_tok, 0, sizeof(recv_tok));

      if (!svc_getargs(rqst->rq_xprt, xdr_rpc_gss_init_args,
                  (caddr_t)&recv_tok))
            return (FALSE);

      gr->gr_major = gss_accept_sec_context(&gr->gr_minor,
                                    &gd->ctx,
                                    svcauth_gss_creds,
                                    &recv_tok,
                                    GSS_C_NO_CHANNEL_BINDINGS,
                                    &gd->client_name,
```

```
                                    &mech,
                                    &gr->gr_token,
                                    &ret_flags,
                                    NULL,
                                    NULL);

      svc_freeargs(rqst->rq_xprt, xdr_rpc_gss_init_args,
(caddr_t)&recv_tok);

      log_status("accept_sec_context", gr->gr_major, gr->gr_minor);
      if (gr->gr_major != GSS_S_COMPLETE &&
          gr->gr_major != GSS_S_CONTINUE_NEEDED) {
            badauth(gr->gr_major, gr->gr_minor, rqst->rq_xprt);
            gd->ctx = GSS_C_NO_CONTEXT;
            goto errout;
      }
      /*
       * ANDROS: krb5 mechglue returns ctx of size 8 - two pointers,
       * one to the mechanism oid, one to the internal_ctx_id
       */
      if ((gr->gr_ctx.value = mem_alloc(sizeof(gss_union_ctx_id_desc)))
== NULL) {
            fprintf(stderr, "svcauth_gss_accept_context: out of
memory\n");
            goto errout;
      }
      memcpy(gr->gr_ctx.value, gd->ctx, sizeof(gss_union_ctx_id_desc));
      gr->gr_ctx.length = sizeof(gss_union_ctx_id_desc);

      /* gr->gr_win = 0x00000005; ANDROS: for debugging linux kernel
version...  */
      gr->gr_win = sizeof(gd->seqmask) * 8;

      /* Save client info. */
      gd->sec.mech = mech;
      gd->sec.qop = GSS_C_QOP_DEFAULT;
      gd->sec.svc = gc->gc_svc;
      gd->seq = gc->gc_seq;
      gd->win = gr->gr_win;

      if (gr->gr_major == GSS_S_COMPLETE) {
#ifdef SPKM
            /* spkm3: no src_name (anonymous) */
            if(!g_OID_equal(gss_mech_spkm3, mech)) {
#endif
                maj_stat = gss_display_name(&min_stat, gd->client_name,
                                  &gd->cname, &gd->sec.mech);
#ifdef SPKM
            }
#endif
            if (maj_stat != GSS_S_COMPLETE) {
                  log_status("display_name", maj_stat, min_stat);
                  goto errout;
            }
```

```
#ifdef DEBUG
#ifdef HAVE_HEIMDAL
            log_debug("accepted context for %.*s with "
                    "<mech {}, qop %d, svc %d>",
                    gd->cname.length, (char *)gd->cname.value,
                    gd->sec.qop, gd->sec.svc);
#else
            {
                  gss_buffer_desc mechname;

                  gss_oid_to_str(&min_stat, mech, &mechname);

                  log_debug("accepted context for %.*s with "
                          "<mech %.*s, qop %d, svc %d>",
                          gd->cname.length, (char *)gd->cname.value,
                          mechname.length, (char *)mechname.value,
                          gd->sec.qop, gd->sec.svc);

                  gss_release_buffer(&min_stat, &mechname);
            }
#endif
#endif /* DEBUG */
            seq = htonl(gr->gr_win);
            seqbuf.value = &seq;
            seqbuf.length = sizeof(seq);

            gss_release_buffer(&min_stat, &gd->checksum);
            maj_stat = gss_sign(&min_stat, gd->ctx, GSS_C_QOP_DEFAULT,
                           &seqbuf, &gd->checksum);

            if (maj_stat != GSS_S_COMPLETE) {
                  goto errout;
            }


            rqst->rq_xprt->xp_verf.oa_flavor = RPCSEC_GSS;
            rqst->rq_xprt->xp_verf.oa_base = gd->checksum.value;
            rqst->rq_xprt->xp_verf.oa_length = gd->checksum.length;
      }
      return (TRUE);
errout:
      gss_release_buffer(&min_stat, &gr->gr_token);
      return (FALSE);
}
<sep>
  bool remoteComplete() const { return state_.remote_complete_; }
<sep>
static int intel_engine_setup(struct intel_gt *gt, enum intel_engine_id
id)
{
      const struct engine_info *info = &intel_engines[id];
      struct drm_i915_private *i915 = gt->i915;
      struct intel_engine_cs *engine;
```

```
	BUILD_BUG_ON(MAX_ENGINE_CLASS >= BIT(GEN11_ENGINE_CLASS_WIDTH));
	BUILD_BUG_ON(MAX_ENGINE_INSTANCE >=
BIT(GEN11_ENGINE_INSTANCE_WIDTH));

	if (GEM_DEBUG_WARN_ON(id >= ARRAY_SIZE(gt->engine)))
		return -EINVAL;

	if (GEM_DEBUG_WARN_ON(info->class > MAX_ENGINE_CLASS))
		return -EINVAL;

	if (GEM_DEBUG_WARN_ON(info->instance > MAX_ENGINE_INSTANCE))
		return -EINVAL;

	if (GEM_DEBUG_WARN_ON(gt->engine_class[info->class][info-
>instance]))
		return -EINVAL;

	engine = kzalloc(sizeof(*engine), GFP_KERNEL);
	if (!engine)
		return -ENOMEM;

	BUILD_BUG_ON(BITS_PER_TYPE(engine->mask) < I915_NUM_ENGINES);

	engine->id = id;
	engine->legacy_idx = INVALID_ENGINE;
	engine->mask = BIT(id);
	engine->i915 = i915;
	engine->gt = gt;
	engine->uncore = gt->uncore;
	engine->hw_id = engine->guc_id = info->hw_id;
	engine->mmio_base = __engine_mmio_base(i915, info->mmio_bases);

	engine->class = info->class;
	engine->instance = info->instance;
	__sprint_engine_name(engine);

	engine->props.heartbeat_interval_ms =
		CONFIG_DRM_I915_HEARTBEAT_INTERVAL;
	engine->props.max_busywait_duration_ns =
		CONFIG_DRM_I915_MAX_REQUEST_BUSYWAIT;
	engine->props.preempt_timeout_ms =
		CONFIG_DRM_I915_PREEMPT_TIMEOUT;
	engine->props.stop_timeout_ms =
		CONFIG_DRM_I915_STOP_TIMEOUT;
	engine->props.timeslice_duration_ms =
		CONFIG_DRM_I915_TIMESLICE_DURATION;

	/* Override to uninterruptible for OpenCL workloads. */
	if (INTEL_GEN(i915) == 12 && engine->class == RENDER_CLASS)
		engine->props.preempt_timeout_ms = 0;

	engine->defaults = engine->props; /* never to change again */
```

```
      engine->context_size = intel_engine_context_size(gt, engine-
>class);
      if (WARN_ON(engine->context_size > BIT(20)))
            engine->context_size = 0;
      if (engine->context_size)
            DRIVER_CAPS(i915)->has_logical_contexts = true;

      /* Nothing to do here, execute in order of dependencies */
      engine->schedule = NULL;

      ewma__engine_latency_init(&engine->latency);
      seqlock_init(&engine->stats.lock);

      ATOMIC_INIT_NOTIFIER_HEAD(&engine->context_status_notifier);

      /* Scrub mmio state on takeover */
      intel_engine_sanitize_mmio(engine);

      gt->engine_class[info->class][info->instance] = engine;
      gt->engine[id] = engine;

      return 0;
}
<sep>
QPDFObjectHandle::rotatePage(int angle, bool relative)
{
    if ((angle % 90) != 0)
    {
        throw std::runtime_error(
            "QPDF::rotatePage called with an"
            " angle that is not a multiple of 90");
    }
    int new_angle = angle;
    if (relative)
    {
        int old_angle = 0;
        bool found_rotate = false;
        QPDFObjectHandle cur_obj = *this;
        bool searched_parent = false;
        std::set<QPDFObjGen> visited;
        while (! found_rotate)
        {
            if (visited.count(cur_obj.getObjGen()))
            {
                // Don't get stuck in an infinite loop
                break;
            }
            if (! visited.empty())
            {
                searched_parent = true;
            }
            visited.insert(cur_obj.getObjGen());
            if (cur_obj.getKey("/Rotate").isInteger())
            {
```

```
                found_rotate = true;
                old_angle = cur_obj.getKey("/Rotate").getIntValue();
            }
            else if (cur_obj.getKey("/Parent").isDictionary())
            {
                cur_obj = cur_obj.getKey("/Parent");
            }
            else
            {
                break;
            }
        }
        QTC::TC("qpdf", "QPDFObjectHandle found old angle",
                searched_parent ? 0 : 1);
        if ((old_angle % 90) != 0)
        {
            old_angle = 0;
        }
        new_angle += old_angle;
    }
    new_angle = (new_angle + 360) % 360;
    replaceKey("/Rotate", QPDFObjectHandle::newInteger(new_angle));
}
<sep>
static inline bool can_follow_write_pmd(pmd_t pmd, unsigned int flags)
{
      return pmd_write(pmd) ||
             ((flags & FOLL_FORCE) && (flags & FOLL_COW) &&
pmd_dirty(pmd));
}
<sep>
SYSCALL_DEFINE1(timer_getoverrun, timer_t, timer_id)
{
      struct k_itimer *timr;
      int overrun;
      unsigned long flags;

      timr = lock_timer(timer_id, &flags);
      if (!timr)
            return -EINVAL;

      overrun = timr->it_overrun_last;
      unlock_timer(timr, flags);

      return overrun;
}
<sep>
static pfunc check_literal(struct jv_parser* p) {
  if (p->tokenpos == 0) return 0;

  const char* pattern = 0;
  int plen;
  jv v;
  switch (p->tokenbuf[0]) {
```

```
  case 't': pattern = "true"; plen = 4; v = jv_true(); break;
  case 'f': pattern = "false"; plen = 5; v = jv_false(); break;
  case 'n': pattern = "null"; plen = 4; v = jv_null(); break;
  }
  if (pattern) {
    if (p->tokenpos != plen) return "Invalid literal";
    for (int i=0; i<plen; i++)
      if (p->tokenbuf[i] != pattern[i])
        return "Invalid literal";
    TRY(value(p, v));
  } else {
    // FIXME: better parser
    p->tokenbuf[p->tokenpos] = 0; // FIXME: invalid
    char* end = 0;
    double d = jvp_strtod(&p->dtoa, p->tokenbuf, &end);
    if (end == 0 || *end != 0)
      return "Invalid numeric literal";
    TRY(value(p, jv_number(d)));
  }
  p->tokenpos = 0;
  return 0;
}
<sep>
static int perf_trace_event_perm(struct ftrace_event_call *tp_event,
                         struct perf_event *p_event)
{
      /* The ftrace function trace is allowed only for root. */
      if (ftrace_event_is_function(tp_event) &&
          perf_paranoid_kernel() && !capable(CAP_SYS_ADMIN))
            return -EPERM;

      /* No tracing, just counting, so no obvious leak */
      if (!(p_event->attr.sample_type & PERF_SAMPLE_RAW))
            return 0;

      /* Some events are ok to be traced by non-root users... */
      if (p_event->attach_state == PERF_ATTACH_TASK) {
            if (tp_event->flags & TRACE_EVENT_FL_CAP_ANY)
                  return 0;
      }

      /*
       * ...otherwise raw tracepoint data can be a severe data leak,
       * only allow root to have these.
       */
      if (perf_paranoid_tracepoint_raw() && !capable(CAP_SYS_ADMIN))
            return -EPERM;

      return 0;
}
<sep>
writeRead (const std::string &tempDir,
           const Array2D<unsigned int> &pi,
           const Array2D<half> &ph,
```

```
            const Array2D<float> &pf,
            int W,
            int H,
            LineOrder lorder,
            Compression comp,
          LevelRoundingMode rmode,
            int dx, int dy,
            int xSize, int ySize)
{
    std::string filename = tempDir + "imf_test_scanline_api.exr";

    writeRead (pi, ph, pf, filename.c_str(), lorder, W, H,
               xSize, ySize, dx, dy, comp, ONE_LEVEL, rmode);
    writeRead (pi, ph, pf, filename.c_str(), lorder, W, H,
               xSize, ySize, dx, dy, comp, MIPMAP_LEVELS, rmode);
    writeRead (pi, ph, pf, filename.c_str(), lorder, W, H,
               xSize, ySize, dx, dy, comp, RIPMAP_LEVELS, rmode);
}
<sep>
MagickPrivate void XColorBrowserWidget(Display *display,XWindows
*windows,
  const char *action,char *reply)
{
#define CancelButtonText  "Cancel"
#define ColornameText  "Name:"
#define ColorPatternText  "Pattern:"
#define GrabButtonText  "Grab"
#define ResetButtonText  "Reset"

  char
    **colorlist,
    primary_selection[MagickPathExtent],
    reset_pattern[MagickPathExtent],
    text[MagickPathExtent];

  ExceptionInfo
    *exception;

  int
    x,
    y;

  int
    i;

  static char
    glob_pattern[MagickPathExtent] = "*";

  static MagickStatusType
    mask = (MagickStatusType) (CWWidth | CWHeight | CWX | CWY);

  Status
    status;
```

```
unsigned int
  height,
  text_width,
  visible_colors,
  width;

size_t
  colors,
  delay,
  state;

XColor
  color;

XEvent
  event;

XFontStruct
  *font_info;

XTextProperty
  window_name;

XWidgetInfo
  action_info,
  cancel_info,
  expose_info,
  grab_info,
  list_info,
  mode_info,
  north_info,
  reply_info,
  reset_info,
  scroll_info,
  selection_info,
  slider_info,
  south_info,
  text_info;

XWindowChanges
  window_changes;

/*
  Get color list and sort in ascending order.
*/
assert(display != (Display *) NULL);
assert(windows != (XWindows *) NULL);
assert(action != (char *) NULL);
assert(reply != (char *) NULL);
(void) LogMagickEvent(TraceEvent,GetMagickModule(),"%s",action);
XSetCursorState(display,windows,MagickTrue);
XCheckRefreshWindows(display,windows);
(void) CopyMagickString(reset_pattern,"*",MagickPathExtent);
exception=AcquireExceptionInfo();
```

```
  colorlist=GetColorList(glob_pattern,&colors,exception);
  if (colorlist == (char **) NULL)
    {
      /*
        Pattern failed, obtain all the colors.
      */
      (void) CopyMagickString(glob_pattern,"*",MagickPathExtent);
      colorlist=GetColorList(glob_pattern,&colors,exception);
      if (colorlist == (char **) NULL)
        {
          XNoticeWidget(display,windows,"Unable to obtain colors names:",
            glob_pattern);
          (void) XDialogWidget(display,windows,action,"Enter color
name:",
            reply);
          return;
        }
    }
  /*
    Determine Color Browser widget attributes.
  */
  font_info=windows->widget.font_info;
  text_width=0;
  for (i=0; i < (int) colors; i++)
    if (WidgetTextWidth(font_info,colorlist[i]) > text_width)
      text_width=WidgetTextWidth(font_info,colorlist[i]);
  width=WidgetTextWidth(font_info,(char *) action);
  if (WidgetTextWidth(font_info,CancelButtonText) > width)
    width=WidgetTextWidth(font_info,CancelButtonText);
  if (WidgetTextWidth(font_info,ResetButtonText) > width)
    width=WidgetTextWidth(font_info,ResetButtonText);
  if (WidgetTextWidth(font_info,GrabButtonText) > width)
    width=WidgetTextWidth(font_info,GrabButtonText);
  width+=QuantumMargin;
  if (WidgetTextWidth(font_info,ColorPatternText) > width)
    width=WidgetTextWidth(font_info,ColorPatternText);
  if (WidgetTextWidth(font_info,ColornameText) > width)
    width=WidgetTextWidth(font_info,ColornameText);
  height=(unsigned int) (font_info->ascent+font_info->descent);
  /*
    Position Color Browser widget.
  */
  windows->widget.width=(unsigned int)
    (width+MagickMin((int) text_width,(int)
MaxTextWidth)+6*QuantumMargin);
  windows->widget.min_width=(unsigned int)
    (width+MinTextWidth+4*QuantumMargin);
  if (windows->widget.width < windows->widget.min_width)
    windows->widget.width=windows->widget.min_width;
  windows->widget.height=(unsigned int)
    ((81*height) >> 2)+((13*QuantumMargin) >> 1)+4;
  windows->widget.min_height=(unsigned int)
    (((23*height) >> 1)+((13*QuantumMargin) >> 1)+4);
  if (windows->widget.height < windows->widget.min_height)
```

```
    windows->widget.height=windows->widget.min_height;
  XConstrainWindowPosition(display,&windows->widget);
  /*
    Map Color Browser widget.
  */
  (void) CopyMagickString(windows->widget.name,"Browse and Select a
Color",
    MagickPathExtent);
  status=XStringListToTextProperty(&windows->widget.name,1,&window_name);
  if (status != False)
    {
      XSetWMName(display,windows->widget.id,&window_name);
      XSetWMIconName(display,windows->widget.id,&window_name);
      (void) XFree((void *) window_name.value);
    }
  window_changes.width=(int) windows->widget.width;
  window_changes.height=(int) windows->widget.height;
  window_changes.x=windows->widget.x;
  window_changes.y=windows->widget.y;
  (void) XReconfigureWMWindow(display,windows->widget.id,windows-
>widget.screen,
    mask,&window_changes);
  (void) XMapRaised(display,windows->widget.id);
  windows->widget.mapped=MagickFalse;
  /*
    Respond to X events.
  */
  XGetWidgetInfo((char *) NULL,&mode_info);
  XGetWidgetInfo((char *) NULL,&slider_info);
  XGetWidgetInfo((char *) NULL,&north_info);
  XGetWidgetInfo((char *) NULL,&south_info);
  XGetWidgetInfo((char *) NULL,&expose_info);
  XGetWidgetInfo((char *) NULL,&selection_info);
  visible_colors=0;
  delay=SuspendTime << 2;
  state=UpdateConfigurationState;
  do
  {
    if (state & UpdateConfigurationState)
      {
        int
          id;

        /*
          Initialize button information.
        */
        XGetWidgetInfo(CancelButtonText,&cancel_info);
        cancel_info.width=width;
        cancel_info.height=(unsigned int) ((3*height) >> 1);
        cancel_info.x=(int)
          (windows->widget.width-cancel_info.width-QuantumMargin-2);
        cancel_info.y=(int)
          (windows->widget.height-cancel_info.height-QuantumMargin);
        XGetWidgetInfo(action,&action_info);
```

```
        action_info.width=width;
        action_info.height=(unsigned int) ((3*height) >> 1);
        action_info.x=cancel_info.x-(cancel_info.width+(QuantumMargin >>
1)+
          (action_info.bevel_width << 1));
        action_info.y=cancel_info.y;
        XGetWidgetInfo(GrabButtonText,&grab_info);
        grab_info.width=width;
        grab_info.height=(unsigned int) ((3*height) >> 1);
        grab_info.x=QuantumMargin;
        grab_info.y=((5*QuantumMargin) >> 1)+height;
        XGetWidgetInfo(ResetButtonText,&reset_info);
        reset_info.width=width;
        reset_info.height=(unsigned int) ((3*height) >> 1);
        reset_info.x=QuantumMargin;
        reset_info.y=grab_info.y+grab_info.height+QuantumMargin;
        /*
          Initialize reply information.
        */
        XGetWidgetInfo(reply,&reply_info);
        reply_info.raised=MagickFalse;
        reply_info.bevel_width--;
        reply_info.width=windows->widget.width-width-((6*QuantumMargin)
>> 1);
        reply_info.height=height << 1;
        reply_info.x=(int) (width+(QuantumMargin << 1));
        reply_info.y=action_info.y-reply_info.height-QuantumMargin;
        /*
          Initialize mode information.
        */
        XGetWidgetInfo((char *) NULL,&mode_info);
        mode_info.active=MagickTrue;
        mode_info.bevel_width=0;
        mode_info.width=(unsigned int) (action_info.x-(QuantumMargin <<
1));
        mode_info.height=action_info.height;
        mode_info.x=QuantumMargin;
        mode_info.y=action_info.y;
        /*
          Initialize scroll information.
        */
        XGetWidgetInfo((char *) NULL,&scroll_info);
        scroll_info.bevel_width--;
        scroll_info.width=height;
        scroll_info.height=(unsigned int) (reply_info.y-grab_info.y-
          (QuantumMargin >> 1));
        scroll_info.x=reply_info.x+(reply_info.width-scroll_info.width);
        scroll_info.y=grab_info.y-reply_info.bevel_width;
        scroll_info.raised=MagickFalse;
        scroll_info.trough=MagickTrue;
        north_info=scroll_info;
        north_info.raised=MagickTrue;
        north_info.width-=(north_info.bevel_width << 1);
        north_info.height=north_info.width-1;
```

```
        north_info.x+=north_info.bevel_width;
        north_info.y+=north_info.bevel_width;
        south_info=north_info;
        south_info.y=scroll_info.y+scroll_info.height-
scroll_info.bevel_width-
          south_info.height;
        id=slider_info.id;
        slider_info=north_info;
        slider_info.id=id;
        slider_info.width-=2;

slider_info.min_y=north_info.y+north_info.height+north_info.bevel_width+
          slider_info.bevel_width+2;
        slider_info.height=scroll_info.height-((slider_info.min_y-
          scroll_info.y+1) << 1)+4;
        visible_colors=scroll_info.height/(height+(height >> 3));
        if (colors > visible_colors)
          slider_info.height=(unsigned int)
            ((visible_colors*slider_info.height)/colors);
        slider_info.max_y=south_info.y-south_info.bevel_width-
          slider_info.bevel_width-2;
        slider_info.x=scroll_info.x+slider_info.bevel_width+1;
        slider_info.y=slider_info.min_y;
        expose_info=scroll_info;
        expose_info.y=slider_info.y;
        /*
          Initialize list information.
        */
        XGetWidgetInfo((char *) NULL,&list_info);
        list_info.raised=MagickFalse;
        list_info.bevel_width--;
        list_info.width=(unsigned int)
          (scroll_info.x-reply_info.x-(QuantumMargin >> 1));
        list_info.height=scroll_info.height;
        list_info.x=reply_info.x;
        list_info.y=scroll_info.y;
        if (windows->widget.mapped == MagickFalse)
          state|=JumpListState;
        /*
          Initialize text information.
        */
        *text='\0';
        XGetWidgetInfo(text,&text_info);
        text_info.center=MagickFalse;
        text_info.width=reply_info.width;
        text_info.height=height;
        text_info.x=list_info.x-(QuantumMargin >> 1);
        text_info.y=QuantumMargin;
        /*
          Initialize selection information.
        */
        XGetWidgetInfo((char *) NULL,&selection_info);
        selection_info.center=MagickFalse;
        selection_info.width=list_info.width;
```

```
        selection_info.height=(unsigned int) ((9*height) >> 3);
        selection_info.x=list_info.x;
        state&=(~UpdateConfigurationState);
      }
    if (state & RedrawWidgetState)
      {
        /*
          Redraw Color Browser window.
        */
        x=QuantumMargin;
        y=text_info.y+((text_info.height-height) >> 1)+font_info->ascent;
        (void) XDrawString(display,windows->widget.id,
          windows->widget.annotate_context,x,y,ColorPatternText,
          Extent(ColorPatternText));
        (void)
CopyMagickString(text_info.text,glob_pattern,MagickPathExtent);
        XDrawWidgetText(display,&windows->widget,&text_info);
        XDrawBeveledButton(display,&windows->widget,&grab_info);
        XDrawBeveledButton(display,&windows->widget,&reset_info);
        XDrawBeveledMatte(display,&windows->widget,&list_info);
        XDrawBeveledMatte(display,&windows->widget,&scroll_info);
        XDrawTriangleNorth(display,&windows->widget,&north_info);
        XDrawBeveledButton(display,&windows->widget,&slider_info);
        XDrawTriangleSouth(display,&windows->widget,&south_info);
        x=QuantumMargin;
        y=reply_info.y+((reply_info.height-height) >> 1)+font_info-
>ascent;
        (void) XDrawString(display,windows->widget.id,
          windows->widget.annotate_context,x,y,ColornameText,
          Extent(ColornameText));
        XDrawBeveledMatte(display,&windows->widget,&reply_info);
        XDrawMatteText(display,&windows->widget,&reply_info);
        XDrawBeveledButton(display,&windows->widget,&action_info);
        XDrawBeveledButton(display,&windows->widget,&cancel_info);
        XHighlightWidget(display,&windows-
>widget,BorderOffset,BorderOffset);
        selection_info.id=(~0);
        state|=RedrawActionState;
        state|=RedrawListState;
        state&=(~RedrawWidgetState);
      }
    if (state & UpdateListState)
      {
        char
          **checklist;

        size_t
          number_colors;

        status=XParseColor(display,windows->widget.map_info->colormap,
          glob_pattern,&color);
        if ((status != False) || (strchr(glob_pattern,'-') != (char *)
NULL))
          {
```

```
            /*
              Reply is a single color name-- exit.
            */
            (void) CopyMagickString(reply,glob_pattern,MagickPathExtent);
            (void)
CopyMagickString(glob_pattern,reset_pattern,MagickPathExtent);
            action_info.raised=MagickFalse;
            XDrawBeveledButton(display,&windows->widget,&action_info);
            break;
          }
        /*
          Update color list.
        */
        checklist=GetColorList(glob_pattern,&number_colors,exception);
        if (number_colors == 0)
          {
            (void)
CopyMagickString(glob_pattern,reset_pattern,MagickPathExtent);
            (void) XBell(display,0);
          }
        else
          {
            for (i=0; i < (int) colors; i++)
              colorlist[i]=DestroyString(colorlist[i]);
            if (colorlist != (char **) NULL)
              colorlist=(char **) RelinquishMagickMemory(colorlist);
            colorlist=checklist;
            colors=number_colors;
          }
        /*
          Sort color list in ascending order.
        */
        slider_info.height=
          scroll_info.height-((slider_info.min_y-scroll_info.y+1) <<
1)+1;
        if (colors > visible_colors)
          slider_info.height=(unsigned int)
            ((visible_colors*slider_info.height)/colors);
        slider_info.max_y=south_info.y-south_info.bevel_width-
          slider_info.bevel_width-2;
        slider_info.id=0;
        slider_info.y=slider_info.min_y;
        expose_info.y=slider_info.y;
        selection_info.id=(~0);
        list_info.id=(~0);
        state|=RedrawListState;
        /*
          Redraw color name & reply.
        */
        *reply_info.text='\0';
        reply_info.cursor=reply_info.text;
        (void)
CopyMagickString(text_info.text,glob_pattern,MagickPathExtent);
        XDrawWidgetText(display,&windows->widget,&text_info);
```

```
        XDrawMatteText(display,&windows->widget,&reply_info);
        XDrawBeveledMatte(display,&windows->widget,&scroll_info);
        XDrawTriangleNorth(display,&windows->widget,&north_info);
        XDrawBeveledButton(display,&windows->widget,&slider_info);
        XDrawTriangleSouth(display,&windows->widget,&south_info);
        XHighlightWidget(display,&windows-
>widget,BorderOffset,BorderOffset);
        state&=(~UpdateListState);
      }
    if (state & JumpListState)
      {
        /*
          Jump scroll to match user color.
        */
        list_info.id=(~0);
        for (i=0; i < (int) colors; i++)
          if (LocaleCompare(colorlist[i],reply) >= 0)
            {
              list_info.id=LocaleCompare(colorlist[i],reply) == 0 ? i :
~0;
              break;
            }
        if ((i < slider_info.id) ||
            (i >= (int) (slider_info.id+visible_colors)))
          slider_info.id=i-(visible_colors >> 1);
        selection_info.id=(~0);
        state|=RedrawListState;
        state&=(~JumpListState);
      }
    if (state & RedrawListState)
      {
        /*
          Determine slider id and position.
        */
        if (slider_info.id >= (int) (colors-visible_colors))
          slider_info.id=(int) (colors-visible_colors);
        if ((slider_info.id < 0) || (colors <= visible_colors))
          slider_info.id=0;
        slider_info.y=slider_info.min_y;
        if (colors != 0)
          slider_info.y+=((ssize_t) slider_info.id*(slider_info.max_y-
            slider_info.min_y+1)/colors);
        if (slider_info.id != selection_info.id)
          {
            /*
              Redraw scroll bar and file names.
            */
            selection_info.id=slider_info.id;
            selection_info.y=list_info.y+(height >> 3)+2;
            for (i=0; i < (int) visible_colors; i++)
            {
              selection_info.raised=(slider_info.id+i) != list_info.id ?
                MagickTrue : MagickFalse;
              selection_info.text=(char *) NULL;
```

```
            if ((slider_info.id+i) < (int) colors)
              selection_info.text=colorlist[slider_info.id+i];
            XDrawWidgetText(display,&windows->widget,&selection_info);
            selection_info.y+=(int) selection_info.height;
          }
          /*
            Update slider.
          */
          if (slider_info.y > expose_info.y)
            {
              expose_info.height=(unsigned int) slider_info.y-
expose_info.y;
              expose_info.y=slider_info.y-expose_info.height-
                slider_info.bevel_width-1;
            }
          else
            {
              expose_info.height=(unsigned int) expose_info.y-
slider_info.y;
              expose_info.y=slider_info.y+slider_info.height+
                slider_info.bevel_width+1;
            }
          XDrawTriangleNorth(display,&windows->widget,&north_info);
          XDrawMatte(display,&windows->widget,&expose_info);
          XDrawBeveledButton(display,&windows->widget,&slider_info);
          XDrawTriangleSouth(display,&windows->widget,&south_info);
          expose_info.y=slider_info.y;
        }
      state&=(~RedrawListState);
    }
  if (state & RedrawActionState)
    {
      static char
        colorname[MagickPathExtent];

      /*
        Display the selected color in a drawing area.
      */
      color=windows->widget.pixel_info->matte_color;
      (void) XParseColor(display,windows->widget.map_info->colormap,
        reply_info.text,&windows->widget.pixel_info->matte_color);
      XBestPixel(display,windows->widget.map_info->colormap,(XColor *)
NULL,
        (unsigned int) windows->widget.visual_info->colormap_size,
        &windows->widget.pixel_info->matte_color);
      mode_info.text=colorname;
      (void) FormatLocaleString(mode_info.text,MagickPathExtent,
        "#%02x%02x%02x",windows->widget.pixel_info->matte_color.red,
        windows->widget.pixel_info->matte_color.green,
        windows->widget.pixel_info->matte_color.blue);
      XDrawBeveledButton(display,&windows->widget,&mode_info);
      windows->widget.pixel_info->matte_color=color;
      state&=(~RedrawActionState);
    }
```

```
    /*
      Wait for next event.
    */
    if (north_info.raised && south_info.raised)
      (void) XIfEvent(display,&event,XScreenEvent,(char *) windows);
    else
      {
        /*
          Brief delay before advancing scroll bar.
        */
        XDelay(display,delay);
        delay=SuspendTime;
        (void) XCheckIfEvent(display,&event,XScreenEvent,(char *)
windows);
        if (north_info.raised == MagickFalse)
          if (slider_info.id > 0)
            {
              /*
                Move slider up.
              */
              slider_info.id--;
              state|=RedrawListState;
            }
        if (south_info.raised == MagickFalse)
          if (slider_info.id < (int) colors)
            {
              /*
                Move slider down.
              */
              slider_info.id++;
              state|=RedrawListState;
            }
        if (event.type != ButtonRelease)
          continue;
      }
    switch (event.type)
    {
      case ButtonPress:
      {
        if (MatteIsActive(slider_info,event.xbutton))
          {
            /*
              Track slider.
            */
            slider_info.active=MagickTrue;
            break;
          }
        if (MatteIsActive(north_info,event.xbutton))
          if (slider_info.id > 0)
            {
              /*
                Move slider up.
              */
              north_info.raised=MagickFalse;
```

```
            slider_info.id--;
            state|=RedrawListState;
            break;
          }
        if (MatteIsActive(south_info,event.xbutton))
          if (slider_info.id < (int) colors)
            {
              /*
                Move slider down.
              */
              south_info.raised=MagickFalse;
              slider_info.id++;
              state|=RedrawListState;
              break;
            }
        if (MatteIsActive(scroll_info,event.xbutton))
          {
            /*
              Move slider.
            */
            if (event.xbutton.y < slider_info.y)
              slider_info.id-=(visible_colors-1);
            else
              slider_info.id+=(visible_colors-1);
            state|=RedrawListState;
            break;
          }
        if (MatteIsActive(list_info,event.xbutton))
          {
            int
              id;

            /*
              User pressed list matte.
            */
            id=slider_info.id+(event.xbutton.y-(list_info.y+(height >>
1))+1)/
              selection_info.height;
            if (id >= (int) colors)
              break;
            (void) CopyMagickString(reply_info.text,colorlist[id],
              MagickPathExtent);
            reply_info.highlight=MagickFalse;
            reply_info.marker=reply_info.text;
            reply_info.cursor=reply_info.text+Extent(reply_info.text);
            XDrawMatteText(display,&windows->widget,&reply_info);
            state|=RedrawActionState;
            if (id == list_info.id)
              {
                (void) CopyMagickString(glob_pattern,reply_info.text,
                  MagickPathExtent);
                state|=UpdateListState;
              }
            selection_info.id=(~0);
```

```
            list_info.id=id;
            state|=RedrawListState;
            break;
          }
        if (MatteIsActive(grab_info,event.xbutton))
          {
            /*
              User pressed Grab button.
            */
            grab_info.raised=MagickFalse;
            XDrawBeveledButton(display,&windows->widget,&grab_info);
            break;
          }
        if (MatteIsActive(reset_info,event.xbutton))
          {
            /*
              User pressed Reset button.
            */
            reset_info.raised=MagickFalse;
            XDrawBeveledButton(display,&windows->widget,&reset_info);
            break;
          }
        if (MatteIsActive(mode_info,event.xbutton))
          {
            /*
              User pressed mode button.
            */
            if (mode_info.text != (char *) NULL)
              (void) CopyMagickString(reply_info.text,mode_info.text,
                MagickPathExtent);
            (void) CopyMagickString(primary_selection,reply_info.text,
              MagickPathExtent);
            (void) XSetSelectionOwner(display,XA_PRIMARY,windows-
>widget.id,
              event.xbutton.time);
            reply_info.highlight=XGetSelectionOwner(display,XA_PRIMARY)
==
              windows->widget.id ? MagickTrue : MagickFalse;
            reply_info.marker=reply_info.text;
            reply_info.cursor=reply_info.text+Extent(reply_info.text);
            XDrawMatteText(display,&windows->widget,&reply_info);
            break;
          }
        if (MatteIsActive(action_info,event.xbutton))
          {
            /*
              User pressed action button.
            */
            action_info.raised=MagickFalse;
            XDrawBeveledButton(display,&windows->widget,&action_info);
            break;
          }
        if (MatteIsActive(cancel_info,event.xbutton))
          {
```

```
            /*
              User pressed Cancel button.
            */
            cancel_info.raised=MagickFalse;
            XDrawBeveledButton(display,&windows->widget,&cancel_info);
            break;
          }
        if (MatteIsActive(reply_info,event.xbutton) == MagickFalse)
          break;
        if (event.xbutton.button != Button2)
          {
            static Time
              click_time;

            /*
              Move text cursor to position of button press.
            */
            x=event.xbutton.x-reply_info.x-(QuantumMargin >> 2);
            for (i=1; i <= Extent(reply_info.marker); i++)
              if (XTextWidth(font_info,reply_info.marker,i) > x)
                break;
            reply_info.cursor=reply_info.marker+i-1;
            if (event.xbutton.time > (click_time+DoubleClick))
              reply_info.highlight=MagickFalse;
            else
              {
                /*
                  Become the XA_PRIMARY selection owner.
                */
                (void)
CopyMagickString(primary_selection,reply_info.text,
                  MagickPathExtent);
                (void) XSetSelectionOwner(display,XA_PRIMARY,windows-
>widget.id,
                  event.xbutton.time);

reply_info.highlight=XGetSelectionOwner(display,XA_PRIMARY) ==
                  windows->widget.id ? MagickTrue : MagickFalse;
              }
            XDrawMatteText(display,&windows->widget,&reply_info);
            click_time=event.xbutton.time;
            break;
          }
        /*
          Request primary selection.
        */
        (void) XConvertSelection(display,XA_PRIMARY,XA_STRING,XA_STRING,
          windows->widget.id,event.xbutton.time);
        break;
      }
      case ButtonRelease:
      {
        if (windows->widget.mapped == MagickFalse)
          break;
```

```
        if (north_info.raised == MagickFalse)
          {
            /*
              User released up button.
            */
            delay=SuspendTime << 2;
            north_info.raised=MagickTrue;
            XDrawTriangleNorth(display,&windows->widget,&north_info);
          }
        if (south_info.raised == MagickFalse)
          {
            /*
              User released down button.
            */
            delay=SuspendTime << 2;
            south_info.raised=MagickTrue;
            XDrawTriangleSouth(display,&windows->widget,&south_info);
          }
        if (slider_info.active)
          {
            /*
              Stop tracking slider.
            */
            slider_info.active=MagickFalse;
            break;
          }
        if (grab_info.raised == MagickFalse)
          {
            if (event.xbutton.window == windows->widget.id)
              if (MatteIsActive(grab_info,event.xbutton))
                {
                  /*
                    Select a fill color from the X server.
                  */
                  (void) XGetWindowColor(display,windows,reply_info.text,
                    exception);
                  reply_info.marker=reply_info.text;

reply_info.cursor=reply_info.text+Extent(reply_info.text);
                  XDrawMatteText(display,&windows->widget,&reply_info);
                  state|=RedrawActionState;
                }
            grab_info.raised=MagickTrue;
            XDrawBeveledButton(display,&windows->widget,&grab_info);
          }
        if (reset_info.raised == MagickFalse)
          {
            if (event.xbutton.window == windows->widget.id)
              if (MatteIsActive(reset_info,event.xbutton))
                {
                  (void) CopyMagickString(glob_pattern,reset_pattern,
                    MagickPathExtent);
                  state|=UpdateListState;
                }
```

```
            reset_info.raised=MagickTrue;
            XDrawBeveledButton(display,&windows->widget,&reset_info);
          }
        if (action_info.raised == MagickFalse)
          {
            if (event.xbutton.window == windows->widget.id)
              {
                if (MatteIsActive(action_info,event.xbutton))
                  {
                    if (*reply_info.text == '\0')
                      (void) XBell(display,0);
                    else
                      state|=ExitState;
                  }
              }
            action_info.raised=MagickTrue;
            XDrawBeveledButton(display,&windows->widget,&action_info);
          }
        if (cancel_info.raised == MagickFalse)
          {
            if (event.xbutton.window == windows->widget.id)
              if (MatteIsActive(cancel_info,event.xbutton))
                {
                  *reply_info.text='\0';
                  state|=ExitState;
                }
            cancel_info.raised=MagickTrue;
            XDrawBeveledButton(display,&windows->widget,&cancel_info);
          }
        if (MatteIsActive(reply_info,event.xbutton) == MagickFalse)
          break;
        break;
      }
      case ClientMessage:
      {
        /*
          If client window delete message, exit.
        */
        if (event.xclient.message_type != windows->wm_protocols)
          break;
        if (*event.xclient.data.l == (int) windows->wm_take_focus)
          {
            (void)
XSetInputFocus(display,event.xclient.window,RevertToParent,
              (Time) event.xclient.data.l[1]);
            break;
          }
        if (*event.xclient.data.l != (int) windows->wm_delete_window)
          break;
        if (event.xclient.window == windows->widget.id)
          {
            *reply_info.text='\0';
            state|=ExitState;
            break;
```

```
          }
        break;
      }
      case ConfigureNotify:
      {
        /*
          Update widget configuration.
        */
        if (event.xconfigure.window != windows->widget.id)
          break;
        if ((event.xconfigure.width == (int) windows->widget.width) &&
            (event.xconfigure.height == (int) windows->widget.height))
          break;
        windows->widget.width=(unsigned int)
          MagickMax(event.xconfigure.width,(int) windows-
>widget.min_width);
        windows->widget.height=(unsigned int)
          MagickMax(event.xconfigure.height,(int) windows-
>widget.min_height);
        state|=UpdateConfigurationState;
        break;
      }
      case EnterNotify:
      {
        if (event.xcrossing.window != windows->widget.id)
          break;
        state&=(~InactiveWidgetState);
        break;
      }
      case Expose:
      {
        if (event.xexpose.window != windows->widget.id)
          break;
        if (event.xexpose.count != 0)
          break;
        state|=RedrawWidgetState;
        break;
      }
      case KeyPress:
      {
        static char
          command[MagickPathExtent];

        static int
          length;

        static KeySym
          key_symbol;

        /*
          Respond to a user key press.
        */
        if (event.xkey.window != windows->widget.id)
          break;
```

```
length=XLookupString((XKeyEvent *) &event.xkey,command,
  (int) sizeof(command),&key_symbol,(XComposeStatus *) NULL);
*(command+length)='\0';
if (AreaIsActive(scroll_info,event.xkey))
  {
    /*
      Move slider.
    */
    switch ((int) key_symbol)
    {
      case XK_Home:
      case XK_KP_Home:
      {
        slider_info.id=0;
        break;
      }
      case XK_Up:
      case XK_KP_Up:
      {
        slider_info.id--;
        break;
      }
      case XK_Down:
      case XK_KP_Down:
      {
        slider_info.id++;
        break;
      }
      case XK_Prior:
      case XK_KP_Prior:
      {
        slider_info.id-=visible_colors;
        break;
      }
      case XK_Next:
      case XK_KP_Next:
      {
        slider_info.id+=visible_colors;
        break;
      }
      case XK_End:
      case XK_KP_End:
      {
        slider_info.id=(int) colors;
        break;
      }
    }
    state|=RedrawListState;
    break;
  }
if ((key_symbol == XK_Return) || (key_symbol == XK_KP_Enter))
  {
    /*
      Read new color or glob patterm.
```

```
            */
            if (*reply_info.text == '\0')
              break;
            (void)
CopyMagickString(glob_pattern,reply_info.text,MagickPathExtent);
            state|=UpdateListState;
            break;
          }
        if (key_symbol == XK_Control_L)
          {
            state|=ControlState;
            break;
          }
        if (state & ControlState)
          switch ((int) key_symbol)
          {
            case XK_u:
            case XK_U:
            {
              /*
                Erase the entire line of text.
              */
              *reply_info.text='\0';
              reply_info.cursor=reply_info.text;
              reply_info.marker=reply_info.text;
              reply_info.highlight=MagickFalse;
              break;
            }
            default:
              break;
          }
        XEditText(display,&reply_info,key_symbol,command,state);
        XDrawMatteText(display,&windows->widget,&reply_info);
        state|=JumpListState;
        status=XParseColor(display,windows->widget.map_info->colormap,
          reply_info.text,&color);
        if (status != False)
          state|=RedrawActionState;
        break;
      }
      case KeyRelease:
      {
        static char
          command[MagickPathExtent];

        static KeySym
          key_symbol;

        /*
          Respond to a user key release.
        */
        if (event.xkey.window != windows->widget.id)
          break;
        (void) XLookupString((XKeyEvent *) &event.xkey,command,
```

```
            (int) sizeof(command),&key_symbol,(XComposeStatus *) NULL);
          if (key_symbol == XK_Control_L)
            state&=(~ControlState);
          break;
        }
        case LeaveNotify:
        {
          if (event.xcrossing.window != windows->widget.id)
            break;
          state|=InactiveWidgetState;
          break;
        }
        case MapNotify:
        {
          mask&=(~CWX);
          mask&=(~CWY);
          break;
        }
        case MotionNotify:
        {
          /*
            Discard pending button motion events.
          */
          while (XCheckMaskEvent(display,ButtonMotionMask,&event)) ;
          if (slider_info.active)
            {
              /*
                Move slider matte.
              */
              slider_info.y=event.xmotion.y-
                ((slider_info.height+slider_info.bevel_width) >> 1)+1;
              if (slider_info.y < slider_info.min_y)
                slider_info.y=slider_info.min_y;
              if (slider_info.y > slider_info.max_y)
                slider_info.y=slider_info.max_y;
              slider_info.id=0;
              if (slider_info.y != slider_info.min_y)
                slider_info.id=(int) ((colors*(slider_info.y-
                  slider_info.min_y+1))/(slider_info.max_y-
slider_info.min_y+1));
              state|=RedrawListState;
              break;
            }
          if (state & InactiveWidgetState)
            break;
          if (grab_info.raised == MatteIsActive(grab_info,event.xmotion))
            {
              /*
                Grab button status changed.
              */
              grab_info.raised=!grab_info.raised;
              XDrawBeveledButton(display,&windows->widget,&grab_info);
              break;
            }
```

```
        if (reset_info.raised == MatteIsActive(reset_info,event.xmotion))
          {
            /*
              Reset button status changed.
            */
            reset_info.raised=!reset_info.raised;
            XDrawBeveledButton(display,&windows->widget,&reset_info);
            break;
          }
        if (action_info.raised ==
MatteIsActive(action_info,event.xmotion))
          {
            /*
              Action button status changed.
            */
            action_info.raised=action_info.raised == MagickFalse ?
              MagickTrue : MagickFalse;
            XDrawBeveledButton(display,&windows->widget,&action_info);
            break;
          }
        if (cancel_info.raised ==
MatteIsActive(cancel_info,event.xmotion))
          {
            /*
              Cancel button status changed.
            */
            cancel_info.raised=cancel_info.raised == MagickFalse ?
              MagickTrue : MagickFalse;
            XDrawBeveledButton(display,&windows->widget,&cancel_info);
            break;
          }
        break;
      }
      case SelectionClear:
      {
        reply_info.highlight=MagickFalse;
        XDrawMatteText(display,&windows->widget,&reply_info);
        break;
      }
      case SelectionNotify:
      {
        Atom
          type;

        int
          format;

        unsigned char
          *data;

        unsigned long
          after,
          length;
```

```
        /*
          Obtain response from primary selection.
        */
        if (event.xselection.property == (Atom) None)
          break;
        status=XGetWindowProperty(display,event.xselection.requestor,
          event.xselection.property,0L,2047L,MagickTrue,XA_STRING,&type,
          &format,&length,&after,&data);
        if ((status != Success) || (type != XA_STRING) || (format == 32)
||
            (length == 0))
          break;
        if ((Extent(reply_info.text)+length) >= (MagickPathExtent-1))
          (void) XBell(display,0);
        else
          {
            /*
              Insert primary selection in reply text.
            */
            *(data+length)='\0';
            XEditText(display,&reply_info,(KeySym) XK_Insert,(char *)
data,
              state);
            XDrawMatteText(display,&windows->widget,&reply_info);
            state|=JumpListState;
            state|=RedrawActionState;
          }
        (void) XFree((void *) data);
        break;
      }
      case SelectionRequest:
      {
        XSelectionEvent
          notify;

        XSelectionRequestEvent
          *request;

        if (reply_info.highlight == MagickFalse)
          break;
        /*
          Set primary selection.
        */
        request=(&(event.xselectionrequest));
        (void) XChangeProperty(request->display,request->requestor,
          request->property,request->target,8,PropModeReplace,
          (unsigned char *) primary_selection,Extent(primary_selection));
        notify.type=SelectionNotify;
        notify.send_event=MagickTrue;
        notify.display=request->display;
        notify.requestor=request->requestor;
        notify.selection=request->selection;
        notify.target=request->target;
        notify.time=request->time;
```

```
        if (request->property == None)
          notify.property=request->target;
        else
          notify.property=request->property;
        (void) XSendEvent(request->display,request->requestor,False,
          NoEventMask,(XEvent *) &notify);
      }
      default:
        break;
    }
  } while ((state & ExitState) == 0);
  XSetCursorState(display,windows,MagickFalse);
  (void) XWithdrawWindow(display,windows->widget.id,windows-
>widget.screen);
  XCheckRefreshWindows(display,windows);
  /*
    Free color list.
  */
  for (i=0; i < (int) colors; i++)
    colorlist[i]=DestroyString(colorlist[i]);
  if (colorlist != (char **) NULL)
    colorlist=(char **) RelinquishMagickMemory(colorlist);
  exception=DestroyExceptionInfo(exception);
  if ((*reply == '\0') || (strchr(reply,'-') != (char *) NULL))
    return;
  status=XParseColor(display,windows->widget.map_info-
>colormap,reply,&color);
  if (status != False)
    return;
  XNoticeWidget(display,windows,"Color is unknown to X server:",reply);
  (void) CopyMagickString(reply,"gray",MagickPathExtent);
}
<sep>
mail_parser_run (EMailParser *parser,
                 EMailPartList *part_list,
                 GCancellable *cancellable)
{
      EMailExtensionRegistry *reg;
      CamelMimeMessage *message;
      EMailPart *mail_part;
      GQueue *parsers;
      GQueue mail_part_queue = G_QUEUE_INIT;
      GList *iter;
      GString *part_id;

      if (cancellable)
            g_object_ref (cancellable);
      else
            cancellable = g_cancellable_new ();

      g_mutex_lock (&parser->priv->mutex);
      g_hash_table_insert (parser->priv->ongoing_part_lists, cancellable,
part_list);
      g_mutex_unlock (&parser->priv->mutex);
```

```
	message = e_mail_part_list_get_message (part_list);

	reg = e_mail_parser_get_extension_registry (parser);

	parsers = e_mail_extension_registry_get_for_mime_type (
		reg, "application/vnd.evolution.message");

	if (parsers == NULL)
		parsers = e_mail_extension_registry_get_for_mime_type (
			reg, "message/*");

	/* No parsers means the internal Evolution parser
	 * extensions were not loaded. Something is terribly wrong! */
	g_return_if_fail (parsers != NULL);

	part_id = g_string_new (".message");

	mail_part = e_mail_part_new (CAMEL_MIME_PART (message),
".message");
	e_mail_part_list_add_part (part_list, mail_part);
	g_object_unref (mail_part);

	for (iter = parsers->head; iter; iter = iter->next) {
		EMailParserExtension *extension;
		gboolean message_handled;

		if (g_cancellable_is_cancelled (cancellable))
			break;

		extension = iter->data;
		if (!extension)
			continue;

		message_handled = e_mail_parser_extension_parse (
			extension, parser,
			CAMEL_MIME_PART (message),
			part_id, cancellable, &mail_part_queue);

		if (message_handled)
			break;
	}

	while (!g_queue_is_empty (&mail_part_queue)) {
		mail_part = g_queue_pop_head (&mail_part_queue);
		e_mail_part_list_add_part (part_list, mail_part);
		g_object_unref (mail_part);
	}

	g_mutex_lock (&parser->priv->mutex);
	g_hash_table_remove (parser->priv->ongoing_part_lists,
cancellable);
	g_mutex_unlock (&parser->priv->mutex);
```

```
      g_clear_object (&cancellable);
      g_string_free (part_id, TRUE);
}
<sep>
njs_iterator_to_array(njs_vm_t *vm, njs_value_t *iterator)
{
    int64_t              length;
    njs_int_t            ret;
    njs_iterator_args_t  args;

    njs_memzero(&args, sizeof(njs_iterator_args_t));

    ret = njs_object_length(vm, iterator, &length);
    if (njs_slow_path(ret != NJS_OK)) {
        return NULL;
    }

    args.data = njs_array_alloc(vm, 1, length, 0);
    if (njs_slow_path(args.data == NULL)) {
        return NULL;
    }

    args.value = iterator;
    args.to = length;

    ret = njs_object_iterate(vm, &args, njs_iterator_to_array_handler);
    if (njs_slow_path(ret == NJS_ERROR)) {
        njs_mp_free(vm->mem_pool, args.data);
        return NULL;
    }

    return args.data;
}
<sep>
static void sub_remove(struct idr *idp, int shift, int id)
{
      struct idr_layer *p = idp->top;
      struct idr_layer **pa[MAX_IDR_LEVEL];
      struct idr_layer ***paa = &pa[0];
      struct idr_layer *to_free;
      int n;

      *paa = NULL;
      *++paa = &idp->top;

      while ((shift > 0) && p) {
            n = (id >> shift) & IDR_MASK;
            __clear_bit(n, &p->bitmap);
            *++paa = &p->ary[n];
            p = p->ary[n];
            shift -= IDR_BITS;
      }
      n = id & IDR_MASK;
      if (likely(p != NULL && test_bit(n, &p->bitmap))){
```

```
            __clear_bit(n, &p->bitmap);
            rcu_assign_pointer(p->ary[n], NULL);
            to_free = NULL;
            while(*paa && ! --((**paa)->count)){
                  if (to_free)
                        free_layer(to_free);
                  to_free = **paa;
                  **paa-- = NULL;
            }
            if (!*paa)
                  idp->layers = 0;
            if (to_free)
                  free_layer(to_free);
      } else
            idr_remove_warning(id);
}
<sep>
void _moddeinit(module_unload_intent_t intent)
{
      service_named_unbind_command("chanserv", &cs_flags);
}
<sep>
rsvg_state_inherit_run (RsvgState * dst, const RsvgState * src,
                        const InheritanceFunction function, const
gboolean inherituninheritables)
{
    gint i;

    if (function (dst->has_current_color, src->has_current_color))
        dst->current_color = src->current_color;
    if (function (dst->has_flood_color, src->has_flood_color))
        dst->flood_color = src->flood_color;
    if (function (dst->has_flood_opacity, src->has_flood_opacity))
        dst->flood_opacity = src->flood_opacity;
    if (function (dst->has_fill_server, src->has_fill_server)) {
        rsvg_paint_server_ref (src->fill);
        if (dst->fill)
            rsvg_paint_server_unref (dst->fill);
        dst->fill = src->fill;
    }
    if (function (dst->has_fill_opacity, src->has_fill_opacity))
        dst->fill_opacity = src->fill_opacity;
    if (function (dst->has_fill_rule, src->has_fill_rule))
        dst->fill_rule = src->fill_rule;
    if (function (dst->has_clip_rule, src->has_clip_rule))
        dst->clip_rule = src->clip_rule;
    if (function (dst->overflow, src->overflow))
        dst->overflow = src->overflow;
    if (function (dst->has_stroke_server, src->has_stroke_server)) {
        rsvg_paint_server_ref (src->stroke);
        if (dst->stroke)
            rsvg_paint_server_unref (dst->stroke);
        dst->stroke = src->stroke;
    }
```

```
    if (function (dst->has_stroke_opacity, src->has_stroke_opacity))
        dst->stroke_opacity = src->stroke_opacity;
    if (function (dst->has_stroke_width, src->has_stroke_width))
        dst->stroke_width = src->stroke_width;
    if (function (dst->has_miter_limit, src->has_miter_limit))
        dst->miter_limit = src->miter_limit;
    if (function (dst->has_cap, src->has_cap))
        dst->cap = src->cap;
    if (function (dst->has_join, src->has_join))
        dst->join = src->join;
    if (function (dst->has_stop_color, src->has_stop_color))
        dst->stop_color = src->stop_color;
    if (function (dst->has_stop_opacity, src->has_stop_opacity))
        dst->stop_opacity = src->stop_opacity;
    if (function (dst->has_cond, src->has_cond))
        dst->cond_true = src->cond_true;
    if (function (dst->has_font_size, src->has_font_size))
        dst->font_size = src->font_size;
    if (function (dst->has_font_style, src->has_font_style))
        dst->font_style = src->font_style;
    if (function (dst->has_font_variant, src->has_font_variant))
        dst->font_variant = src->font_variant;
    if (function (dst->has_font_weight, src->has_font_weight))
        dst->font_weight = src->font_weight;
    if (function (dst->has_font_stretch, src->has_font_stretch))
        dst->font_stretch = src->font_stretch;
    if (function (dst->has_font_decor, src->has_font_decor))
        dst->font_decor = src->font_decor;
    if (function (dst->has_text_dir, src->has_text_dir))
        dst->text_dir = src->text_dir;
    if (function (dst->has_text_gravity, src->has_text_gravity))
        dst->text_gravity = src->text_gravity;
    if (function (dst->has_unicode_bidi, src->has_unicode_bidi))
        dst->unicode_bidi = src->unicode_bidi;
    if (function (dst->has_text_anchor, src->has_text_anchor))
        dst->text_anchor = src->text_anchor;
    if (function (dst->has_letter_spacing, src->has_letter_spacing))
        dst->letter_spacing = src->letter_spacing;
    if (function (dst->has_startMarker, src->has_startMarker))
        dst->startMarker = src->startMarker;
    if (function (dst->has_middleMarker, src->has_middleMarker))
        dst->middleMarker = src->middleMarker;
    if (function (dst->has_endMarker, src->has_endMarker))
        dst->endMarker = src->endMarker;
      if (function (dst->has_shape_rendering_type, src-
>has_shape_rendering_type))
           dst->shape_rendering_type = src->shape_rendering_type;
      if (function (dst->has_text_rendering_type, src-
>has_text_rendering_type))
           dst->text_rendering_type = src->text_rendering_type;

    if (function (dst->has_font_family, src->has_font_family)) {
        g_free (dst->font_family);      /* font_family is always set to
something */
```

```
        dst->font_family = g_strdup (src->font_family);
    }

    if (function (dst->has_space_preserve, src->has_space_preserve))
        dst->space_preserve = src->space_preserve;

    if (function (dst->has_visible, src->has_visible))
        dst->visible = src->visible;

    if (function (dst->has_lang, src->has_lang)) {
        if (dst->has_lang)
            g_free (dst->lang);
        dst->lang = g_strdup (src->lang);
    }

    if (src->dash.n_dash > 0 && (function (dst->has_dash, src-
>has_dash))) {
        if (dst->has_dash)
            g_free (dst->dash.dash);

        dst->dash.dash = g_new (gdouble, src->dash.n_dash);
        dst->dash.n_dash = src->dash.n_dash;
        for (i = 0; i < src->dash.n_dash; i++)
            dst->dash.dash[i] = src->dash.dash[i];
    }

    if (function (dst->has_dashoffset, src->has_dashoffset)) {
        dst->dash.offset = src->dash.offset;
    }

    if (inherituninheritables) {
        dst->clip_path_ref = src->clip_path_ref;
        dst->mask = src->mask;
        dst->enable_background = src->enable_background;
        dst->adobe_blend = src->adobe_blend;
        dst->opacity = src->opacity;
        dst->filter = src->filter;
        dst->comp_op = src->comp_op;
    }
}
<sep>
void jpc_qmfb_join_row(jpc_fix_t *a, int numcols, int parity)
{

      int bufsize = JPC_CEILDIVPOW2(numcols, 1);
      jpc_fix_t joinbuf[QMFB_JOINBUFSIZE];
      jpc_fix_t *buf = joinbuf;
      register jpc_fix_t *srcptr;
      register jpc_fix_t *dstptr;
      register int n;
      int hstartcol;

      /* Allocate memory for the join buffer from the heap. */
      if (bufsize > QMFB_JOINBUFSIZE) {
```

```
            if (!(buf = jas_malloc(bufsize * sizeof(jpc_fix_t)))) {
                  /* We have no choice but to commit suicide. */
                  abort();
            }
      }

      hstartcol = (numcols + 1 - parity) >> 1;

      /* Save the samples from the lowpass channel. */
      n = hstartcol;
      srcptr = &a[0];
      dstptr = buf;
      while (n-- > 0) {
            *dstptr = *srcptr;
            ++srcptr;
            ++dstptr;
      }
      /* Copy the samples from the highpass channel into place. */
      srcptr = &a[hstartcol];
      dstptr = &a[1 - parity];
      n = numcols - hstartcol;
      while (n-- > 0) {
            *dstptr = *srcptr;
            dstptr += 2;
            ++srcptr;
      }
      /* Copy the samples from the lowpass channel into place. */
      srcptr = buf;
      dstptr = &a[parity];
      n = hstartcol;
      while (n-- > 0) {
            *dstptr = *srcptr;
            dstptr += 2;
            ++srcptr;
      }

      /* If the join buffer was allocated on the heap, free this memory.
*/
      if (buf != joinbuf) {
            jas_free(buf);
      }

}
<sep>
static int check_nonce(request_rec *r, digest_header_rec *resp,
                       const digest_config_rec *conf)
{
    apr_time_t dt;
    time_rec nonce_time;
    char tmp, hash[NONCE_HASH_LEN+1];

    if (strlen(resp->nonce) != NONCE_LEN) {
        ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r, APLOGNO(01775)
                      "invalid nonce %s received - length is not %d",
```

```
                    resp->nonce, NONCE_LEN);
        note_digest_auth_failure(r, conf, resp, 1);
        return HTTP_UNAUTHORIZED;
    }

    tmp = resp->nonce[NONCE_TIME_LEN];
    resp->nonce[NONCE_TIME_LEN] = '\0';
    apr_base64_decode_binary(nonce_time.arr, resp->nonce);
    gen_nonce_hash(hash, resp->nonce, resp->opaque, r->server, conf);
    resp->nonce[NONCE_TIME_LEN] = tmp;
    resp->nonce_time = nonce_time.time;

    if (strcmp(hash, resp->nonce+NONCE_TIME_LEN)) {
        ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r, APLOGNO(01776)
                      "invalid nonce %s received - hash is not %s",
                      resp->nonce, hash);
        note_digest_auth_failure(r, conf, resp, 1);
        return HTTP_UNAUTHORIZED;
    }

    dt = r->request_time - nonce_time.time;
    if (conf->nonce_lifetime > 0 && dt < 0) {
        ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r, APLOGNO(01777)
                      "invalid nonce %s received - user attempted "
                      "time travel", resp->nonce);
        note_digest_auth_failure(r, conf, resp, 1);
        return HTTP_UNAUTHORIZED;
    }

    if (conf->nonce_lifetime > 0) {
        if (dt > conf->nonce_lifetime) {
            ap_log_rerror(APLOG_MARK, APLOG_INFO, 0,r, APLOGNO(01778)
                          "user %s: nonce expired (%.2f seconds old "
                          "- max lifetime %.2f) - sending new nonce",
                          r->user, (double)apr_time_sec(dt),
                          (double)apr_time_sec(conf->nonce_lifetime));
            note_digest_auth_failure(r, conf, resp, 1);
            return HTTP_UNAUTHORIZED;
        }
    }
    else if (conf->nonce_lifetime == 0 && resp->client) {
        if (memcmp(resp->client->last_nonce, resp->nonce, NONCE_LEN)) {
            ap_log_rerror(APLOG_MARK, APLOG_INFO, 0, r, APLOGNO(01779)
                          "user %s: one-time-nonce mismatch - sending "
                          "new nonce", r->user);
            note_digest_auth_failure(r, conf, resp, 1);
            return HTTP_UNAUTHORIZED;
        }
    }
    /* else (lifetime < 0) => never expires */

    return OK;
}
<sep>
```

```
Http::Stream::socketState()
{
    switch (clientStreamStatus(getTail(), http)) {

    case STREAM_NONE:
        /* check for range support ending */
        if (http->request->range) {
            /* check: reply was parsed and range iterator was initialized
*/
            assert(http->range_iter.valid);
            /* filter out data according to range specs */

            if (!canPackMoreRanges()) {
                debugs(33, 5, "Range request at end of returnable " <<
                       "range sequence on " << clientConnection);
                // we got everything we wanted from the store
                return STREAM_COMPLETE;
            }
        } else if (reply && reply->contentRange()) {
            /* reply has content-range, but Squid is not managing ranges
*/
            const int64_t &bytesSent = http->out.offset;
            const int64_t &bytesExpected = reply->contentRange()-
>spec.length;

            debugs(33, 7, "body bytes sent vs. expected: " <<
                   bytesSent << " ? " << bytesExpected << " (+" <<
                   reply->contentRange()->spec.offset << ")");

            // did we get at least what we expected, based on range
specs?

            if (bytesSent == bytesExpected) // got everything
                return STREAM_COMPLETE;

            if (bytesSent > bytesExpected) // Error: Sent more than
expected
                return STREAM_UNPLANNED_COMPLETE;
        }

        return STREAM_NONE;

    case STREAM_COMPLETE:
        return STREAM_COMPLETE;

    case STREAM_UNPLANNED_COMPLETE:
        return STREAM_UNPLANNED_COMPLETE;

    case STREAM_FAILED:
        return STREAM_FAILED;
    }

    fatal ("unreachable code\n");
    return STREAM_NONE;
```

```
}
<sep>
static int huft_build(const unsigned *b, const unsigned n,
                  const unsigned s, const unsigned short *d,
                  const unsigned char *e, huft_t **t, unsigned *m)
{
      unsigned a;             /* counter for codes of length k */
      unsigned c[BMAX + 1];   /* bit length count table */
      unsigned eob_len;       /* length of end-of-block code (value 256)
*/
      unsigned f;             /* i repeats in table every f entries */
      int g;                  /* maximum code length */
      int htl;                /* table level */
      unsigned i;             /* counter, current code */
      unsigned j;             /* counter */
      int k;                  /* number of bits in current code */
      unsigned *p;            /* pointer into c[], b[], or v[] */
      huft_t *q;              /* points to current table */
      huft_t r;               /* table entry for structure assignment */
      huft_t *u[BMAX];        /* table stack */
      unsigned v[N_MAX];      /* values in order of bit length */
      int ws[BMAX + 1];       /* bits decoded stack */
      int w;                  /* bits decoded */
      unsigned x[BMAX + 1];   /* bit offsets, then code stack */
      unsigned *xp;           /* pointer into x */
      int y;                  /* number of dummy codes added */
      unsigned z;             /* number of entries in current table */

      /* Length of EOB code, if any */
      eob_len = n > 256 ? b[256] : BMAX;

      *t = NULL;

      /* Generate counts for each bit length */
      memset(c, 0, sizeof(c));
      p = (unsigned *) b; /* cast allows us to reuse p for pointing to b
*/
      i = n;
      do {
            c[*p]++; /* assume all entries <= BMAX */
            p++;     /* can't combine with above line (Solaris bug) */
      } while (--i);
      if (c[0] == n) {  /* null input - all zero length codes */
            *m = 0;
            return 2;
      }

      /* Find minimum and maximum length, bound *m by those */
      for (j = 1; (j <= BMAX) && (c[j] == 0); j++)
            continue;
      k = j; /* minimum code length */
      for (i = BMAX; (c[i] == 0) && i; i--)
            continue;
      g = i; /* maximum code length */
```

```
      *m = (*m < j) ? j : ((*m > i) ? i : *m);

      /* Adjust last length count to fill out codes, if needed */
      for (y = 1 << j; j < i; j++, y <<= 1) {
            y -= c[j];
            if (y < 0)
                  return 2; /* bad input: more codes than bits */
      }
      y -= c[i];
      if (y < 0)
            return 2;
      c[i] += y;

      /* Generate starting offsets into the value table for each length
*/
      x[1] = j = 0;
      p = c + 1;
      xp = x + 2;
      while (--i) { /* note that i == g from above */
            j += *p++;
            *xp++ = j;
      }

      /* Make a table of values in order of bit lengths */
      p = (unsigned *) b;
      i = 0;
      do {
            j = *p++;
            if (j != 0) {
                  v[x[j]++] = i;
            }
      } while (++i < n);

      /* Generate the Huffman codes and for each, make the table entries
*/
      x[0] = i = 0;   /* first Huffman code is zero */
      p = v;          /* grab values in bit order */
      htl = -1;       /* no tables yet--level -1 */
      w = ws[0] = 0;  /* bits decoded */
      u[0] = NULL;    /* just to keep compilers happy */
      q = NULL;       /* ditto */
      z = 0;          /* ditto */

      /* go through the bit lengths (k already is bits in shortest code)
*/
      for (; k <= g; k++) {
            a = c[k];
            while (a--) {
                  /* here i is the Huffman code of length k bits for value
*p */
                  /* make tables up to required level */
                  while (k > ws[htl + 1]) {
                        w = ws[++htl];
```

```
                        /* compute minimum size table less than or equal
to *m bits */
                        z = g - w;
                        z = z > *m ? *m : z; /* upper limit on table size
*/
                        j = k - w;
                        f = 1 << j;
                        if (f > a + 1) { /* try a k-w bit table */
                              /* too few codes for k-w bit table */
                              f -= a + 1; /* deduct codes from patterns
left */
                              xp = c + k;
                              while (++j < z) { /* try smaller tables up
to z bits */
                                    f <<= 1;
                                    if (f <= *++xp) {
                                          break; /* enough codes to use up
j bits */
                                    }
                                    f -= *xp; /* else deduct codes from
patterns */
                              }
                        }
                        j = (w + j > eob_len && w < eob_len) ? eob_len - w
: j;  /* make EOB code end at table */
                        z = 1 << j; /* table entries for j-bit table */
                        ws[htl+1] = w + j;     /* set bits decoded in
stack */

                        /* allocate and link in new table */
                        q = xzalloc((z + 1) * sizeof(huft_t));
                        *t = q + 1; /* link to list for huft_free() */
                        t = &(q->v.t);
                        u[htl] = ++q;    /* table starts after link */

                        /* connect to last table, if there is one */
                        if (htl) {
                              x[htl] = i; /* save pattern for backing up
*/
                              r.b = (unsigned char) (w - ws[htl - 1]); /*
bits to dump before this table */
                              r.e = (unsigned char) (16 + j); /* bits in
this table */
                              r.v.t = q; /* pointer to this table */
                              j = (i & ((1 << w) - 1)) >> ws[htl - 1];
                              u[htl - 1][j] = r; /* connect to last table
*/
                        }
                  }

                  /* set up table entry in r */
                  r.b = (unsigned char) (k - w);
                  if (p >= v + n) {
                        r.e = 99; /* out of values--invalid code */
```

```
                } else if (*p < s) {
                     r.e = (unsigned char) (*p < 256 ? 16 : 15);   /*
256 is EOB code */
                     r.v.n = (unsigned short) (*p++); /* simple code is
just the value */
                } else {
                     r.e = (unsigned char) e[*p - s]; /* non-simple--
look up in lists */
                     r.v.n = d[*p++ - s];
                }

                /* fill code-like entries with r */
                f = 1 << (k - w);
                for (j = i >> w; j < z; j += f) {
                     q[j] = r;
                }

                /* backwards increment the k-bit code i */
                for (j = 1 << (k - 1); i & j; j >>= 1) {
                     i ^= j;
                }
                i ^= j;

                /* backup over finished tables */
                while ((i & ((1 << w) - 1)) != x[htl]) {
                     w = ws[--htl];
                }
           }
      }

      /* return actual size of base table */
      *m = ws[1];

      /* Return 1 if we were given an incomplete table */
      return y != 0 && g != 1;
}
<sep>
bool parse_vcol_defs(THD *thd, MEM_ROOT *mem_root, TABLE *table,
                     bool *error_reported, vcol_init_mode mode)
{
  CHARSET_INFO *save_character_set_client= thd-
>variables.character_set_client;
  CHARSET_INFO *save_collation= thd->variables.collation_connection;
  Query_arena  *backup_stmt_arena_ptr= thd->stmt_arena;
  const uchar *pos= table->s->vcol_defs.str;
  const uchar *end= pos + table->s->vcol_defs.length;
  Field **field_ptr= table->field - 1;
  Field **vfield_ptr= table->vfield;
  Field **dfield_ptr= table->default_field;
  Virtual_column_info **check_constraint_ptr= table->check_constraints;
  sql_mode_t saved_mode= thd->variables.sql_mode;
  Query_arena backup_arena;
  Virtual_column_info *vcol= 0;
  StringBuffer<MAX_FIELD_WIDTH> expr_str;
```

```
  bool res= 1;
  DBUG_ENTER("parse_vcol_defs");

  if (check_constraint_ptr)
    memcpy(table->check_constraints + table->s->field_check_constraints,
           table->s->check_constraints,
           table->s->table_check_constraints *
sizeof(Virtual_column_info*));

  DBUG_ASSERT(table->expr_arena == NULL);
  /*
    We need to use CONVENTIONAL_EXECUTION here to ensure that
    any new items created by fix_fields() are not reverted.
  */
  table->expr_arena= new (alloc_root(mem_root, sizeof(Table_arena)))
                         Table_arena(mem_root,

Query_arena::STMT_CONVENTIONAL_EXECUTION);
  if (!table->expr_arena)
    DBUG_RETURN(1);

  thd->set_n_backup_active_arena(table->expr_arena, &backup_arena);
  thd->stmt_arena= table->expr_arena;
  thd->update_charset(&my_charset_utf8mb4_general_ci, table->s-
>table_charset);
  expr_str.append(&parse_vcol_keyword);
  thd->variables.sql_mode &= ~MODE_NO_BACKSLASH_ESCAPES;

  while (pos < end)
  {
    uint type, expr_length;
    if (table->s->frm_version >= FRM_VER_EXPRESSSIONS)
    {
      uint field_nr, name_length;
      /* see pack_expression() for how data is stored */
      type= pos[0];
      field_nr= uint2korr(pos+1);
      expr_length= uint2korr(pos+3);
      name_length= pos[5];
      pos+= FRM_VCOL_NEW_HEADER_SIZE + name_length;
      field_ptr= table->field + field_nr;
    }
    else
    {
      /*
        see below in ::init_from_binary_frm_image for how data is stored
        in versions below 10.2 (that includes 5.7 too)
      */
      while (*++field_ptr && !(*field_ptr)->vcol_info) /* no-op */;
      if (!*field_ptr)
      {
        open_table_error(table->s, OPEN_FRM_CORRUPTED, 1);
        goto end;
      }
```

```
      type= (*field_ptr)->vcol_info->stored_in_db
            ? VCOL_GENERATED_STORED : VCOL_GENERATED_VIRTUAL;
      expr_length= uint2korr(pos+1);
      if (table->s->mysql_version > 50700 && table->s->mysql_version <
100000)
        pos+= 4;                          // MySQL from 5.7
      else
        pos+= pos[0] == 2 ? 4 : 3;        // MariaDB from 5.2 to 10.1
    }

    expr_str.length(parse_vcol_keyword.length);
    expr_str.append((char*)pos, expr_length);
    thd->where= vcol_type_name(static_cast<enum_vcol_info_type>(type));

    switch (type) {
    case VCOL_GENERATED_VIRTUAL:
    case VCOL_GENERATED_STORED:
      vcol= unpack_vcol_info_from_frm(thd, mem_root, table, &expr_str,
                                      &((*field_ptr)->vcol_info),
error_reported);
      *(vfield_ptr++)= *field_ptr;
      if (vcol && field_ptr[0]->check_vcol_sql_mode_dependency(thd,
mode))
      {
        DBUG_ASSERT(thd->is_error());
        *error_reported= true;
        goto end;
      }
      break;
    case VCOL_DEFAULT:
      vcol= unpack_vcol_info_from_frm(thd, mem_root, table, &expr_str,
                                      &((*field_ptr)->default_value),
                                      error_reported);
      *(dfield_ptr++)= *field_ptr;
      if (vcol && (vcol->flags & (VCOL_NON_DETERMINISTIC |
VCOL_SESSION_FUNC)))
        table->s->non_determinstic_insert= true;
      break;
    case VCOL_CHECK_FIELD:
      vcol= unpack_vcol_info_from_frm(thd, mem_root, table, &expr_str,
                                      &((*field_ptr)->check_constraint),
                                      error_reported);
      *check_constraint_ptr++= (*field_ptr)->check_constraint;
      break;
    case VCOL_CHECK_TABLE:
      vcol= unpack_vcol_info_from_frm(thd, mem_root, table, &expr_str,
                                      check_constraint_ptr,
error_reported);
      check_constraint_ptr++;
      break;
    }
    if (!vcol)
      goto end;
    pos+= expr_length;
```

```
  }

  /* Now, initialize CURRENT_TIMESTAMP fields */
  for (field_ptr= table->field; *field_ptr; field_ptr++)
  {
    Field *field= *field_ptr;
    if (field->has_default_now_unireg_check())
    {
      expr_str.length(parse_vcol_keyword.length);
      expr_str.append(STRING_WITH_LEN("current_timestamp("));
      expr_str.append_ulonglong(field->decimals());
      expr_str.append(')');
      vcol= unpack_vcol_info_from_frm(thd, mem_root, table, &expr_str,
                                      &((*field_ptr)->default_value),
                                      error_reported);
      *(dfield_ptr++)= *field_ptr;
      if (!field->default_value->expr)
        goto end;
    }
    else if (field->has_update_default_function() && !field-
>default_value)
      *(dfield_ptr++)= *field_ptr;
  }

  if (vfield_ptr)
    *vfield_ptr= 0;

  if (dfield_ptr)
    *dfield_ptr= 0;

  if (check_constraint_ptr)
    *check_constraint_ptr= 0;

  /* Check that expressions aren't referring to not yet initialized
fields */
  for (field_ptr= table->field; *field_ptr; field_ptr++)
  {
    Field *field= *field_ptr;
    if (check_vcol_forward_refs(field, field->vcol_info) ||
        check_vcol_forward_refs(field, field->check_constraint) ||
        check_vcol_forward_refs(field, field->default_value))
    {
      *error_reported= true;
      goto end;
    }
  }

  res=0;
end:
  thd->restore_active_arena(table->expr_arena, &backup_arena);
  thd->stmt_arena= backup_stmt_arena_ptr;
  if (save_character_set_client)
    thd->update_charset(save_character_set_client, save_collation);
  thd->variables.sql_mode= saved_mode;
```

```
  DBUG_RETURN(res);
}
<sep>
static void sixpack_close(struct tty_struct *tty)
{
      struct sixpack *sp;

      write_lock_irq(&disc_data_lock);
      sp = tty->disc_data;
      tty->disc_data = NULL;
      write_unlock_irq(&disc_data_lock);
      if (!sp)
            return;

      /*
       * We have now ensured that nobody can start using ap from now on,
but
       * we have to wait for all existing users to finish.
       */
      if (!refcount_dec_and_test(&sp->refcnt))
            wait_for_completion(&sp->dead);

      /* We must stop the queue to avoid potentially scribbling
       * on the free buffers. The sp->dead completion is not sufficient
       * to protect us from sp->xbuff access.
       */
      netif_stop_queue(sp->dev);

      del_timer_sync(&sp->tx_t);
      del_timer_sync(&sp->resync_t);

      /* Free all 6pack frame buffers. */
      kfree(sp->rbuff);
      kfree(sp->xbuff);

      unregister_netdev(sp->dev);
}
<sep>
static int rdn_name_modify(struct ldb_module *module, struct ldb_request
*req)
{
      struct ldb_context *ldb;
      const struct ldb_val *rdn_val_p;
      struct ldb_message_element *e = NULL;
      struct ldb_control *recalculate_rdn_control = NULL;

      ldb = ldb_module_get_ctx(module);

      /* do not manipulate our control entries */
      if (ldb_dn_is_special(req->op.mod.message->dn)) {
            return ldb_next_request(module, req);
      }

      recalculate_rdn_control = ldb_request_get_control(req,
```

```
                        LDB_CONTROL_RECALCULATE_RDN_OID);
if (recalculate_rdn_control != NULL) {
      struct ldb_message *msg = NULL;
      const char *rdn_name = NULL;
      struct ldb_val rdn_val;
      const struct ldb_schema_attribute *a = NULL;
      struct ldb_request *mod_req = NULL;
      int ret;
      struct ldb_message_element *rdn_del = NULL;
      struct ldb_message_element *name_del = NULL;

      recalculate_rdn_control->critical = false;

      msg = ldb_msg_copy_shallow(req, req->op.mod.message);
      if (msg == NULL) {
            return ldb_module_oom(module);
      }

      /*
       * The caller must pass a dummy 'name' attribute
       * in order to bypass some high level checks.
       *
       * We just remove it and check nothing is left.
       */
      ldb_msg_remove_attr(msg, "name");

      if (msg->num_elements != 0) {
            return ldb_module_operr(module);
      }

      rdn_name = ldb_dn_get_rdn_name(msg->dn);
      if (rdn_name == NULL) {
            return ldb_module_oom(module);
      }

      a = ldb_schema_attribute_by_name(ldb, rdn_name);
      if (a == NULL) {
            return ldb_module_operr(module);
      }

      if (a->name != NULL && strcmp(a->name, "*") != 0) {
            rdn_name = a->name;
      }

      rdn_val_p = ldb_dn_get_rdn_val(msg->dn);
      if (rdn_val_p == NULL) {
            return ldb_module_oom(module);
      }
      rdn_val = ldb_val_dup(msg, rdn_val_p);
      if (rdn_val.length == 0) {
            return ldb_module_oom(module);
      }

      /*
```

```
		 * This is a bit tricky:
		 *
		 * We want _DELETE elements (as "rdn_del" and "name_del"
without
		 * values) first, followed by _ADD (with the real names)
		 * elements (with values). Then we fix up the "rdn_del" and
		 * "name_del" attributes.
		 */

		ret = ldb_msg_add_empty(msg, "rdn_del", LDB_FLAG_MOD_DELETE,
NULL);
		if (ret != 0) {
			return ldb_module_oom(module);
		}
		ret = ldb_msg_add_empty(msg, rdn_name, LDB_FLAG_MOD_ADD,
NULL);
		if (ret != 0) {
			return ldb_module_oom(module);
		}
		ret = ldb_msg_add_value(msg, rdn_name, &rdn_val, NULL);
		if (ret != 0) {
			return ldb_module_oom(module);
		}

		ret = ldb_msg_add_empty(msg, "name_del", LDB_FLAG_MOD_DELETE,
NULL);
		if (ret != 0) {
			return ldb_module_oom(module);
		}
		ret = ldb_msg_add_empty(msg, "name", LDB_FLAG_MOD_ADD, NULL);
		if (ret != 0) {
			return ldb_module_oom(module);
		}
		ret = ldb_msg_add_value(msg, "name", &rdn_val, NULL);
		if (ret != 0) {
			return ldb_module_oom(module);
		}

		rdn_del = ldb_msg_find_element(msg, "rdn_del");
		if (rdn_del == NULL) {
			return ldb_module_operr(module);
		}
		rdn_del->name = talloc_strdup(msg->elements, rdn_name);
		if (rdn_del->name == NULL) {
			return ldb_module_oom(module);
		}
		name_del = ldb_msg_find_element(msg, "name_del");
		if (name_del == NULL) {
			return ldb_module_operr(module);
		}
		name_del->name = talloc_strdup(msg->elements, "name");
		if (name_del->name == NULL) {
			return ldb_module_oom(module);
		}
```

```
		ret = ldb_build_mod_req(&mod_req, ldb,
					req, msg, NULL,
					req, rdn_recalculate_callback,
					req);
		if (ret != LDB_SUCCESS) {
			return ldb_module_done(req, NULL, NULL, ret);
		}
		talloc_steal(mod_req, msg);

		ret = ldb_request_add_control(mod_req,
					      LDB_CONTROL_RECALCULATE_RDN_OID,
					      false, NULL);
		if (ret != LDB_SUCCESS) {
			return ldb_module_done(req, NULL, NULL, ret);
		}
		ret = ldb_request_add_control(mod_req,
					      LDB_CONTROL_PERMISSIVE_MODIFY_OID,
					      false, NULL);
		if (ret != LDB_SUCCESS) {
			return ldb_module_done(req, NULL, NULL, ret);
		}

		/* go on with the call chain */
		return ldb_next_request(module, mod_req);
	}

	rdn_val_p = ldb_dn_get_rdn_val(req->op.mod.message->dn);
	if (rdn_val_p == NULL) {
		return LDB_ERR_OPERATIONS_ERROR;
	}
	if (rdn_val_p->length == 0) {
		ldb_asprintf_errstring(ldb, "Empty RDN value on %s not
permitted!",
				       ldb_dn_get_linearized(req->op.mod.message-
>dn));
		return LDB_ERR_INVALID_DN_SYNTAX;
	}

	e = ldb_msg_find_element(req->op.mod.message, "distinguishedName");
	if (e != NULL) {
		ldb_asprintf_errstring(ldb, "Modify of 'distinguishedName' on
%s not permitted, must use 'rename' operation instead",
				       ldb_dn_get_linearized(req->op.mod.message-
>dn));
		if (LDB_FLAG_MOD_TYPE(e->flags) == LDB_FLAG_MOD_REPLACE) {
			return LDB_ERR_CONSTRAINT_VIOLATION;
		} else {
			return LDB_ERR_UNWILLING_TO_PERFORM;
		}
	}

	if (ldb_msg_find_element(req->op.mod.message, "name")) {
```

```
            ldb_asprintf_errstring(ldb, "Modify of 'name' on %s not
permitted, must use 'rename' operation instead",
                                   ldb_dn_get_linearized(req->op.mod.message-
>dn));
            return LDB_ERR_NOT_ALLOWED_ON_RDN;
      }

      if (ldb_msg_find_element(req->op.mod.message,
ldb_dn_get_rdn_name(req->op.mod.message->dn))) {
            ldb_asprintf_errstring(ldb, "Modify of RDN '%s' on %s not
permitted, must use 'rename' operation instead",
                                   ldb_dn_get_rdn_name(req->op.mod.message-
>dn), ldb_dn_get_linearized(req->op.mod.message->dn));
            return LDB_ERR_NOT_ALLOWED_ON_RDN;
      }

      /* All OK, they kept their fingers out of the special attributes */
      return ldb_next_request(module, req);
}
<sep>
static inline unsigned short ScaleQuantumToShort(const Quantum quantum)
{
#if !defined(MAGICKCORE_HDRI_SUPPORT)
  return((unsigned short) (257UL*quantum));
#else
  if (quantum <= 0.0)
    return(0);
  if ((257.0*quantum) >= 65535.0)
    return(65535);
  return((unsigned short) (257.0*quantum+0.5));
#endif
}
<sep>
plpgsql_validator(PG_FUNCTION_ARGS)
{
      Oid               funcoid = PG_GETARG_OID(0);
      HeapTuple   tuple;
      Form_pg_proc proc;
      char        functyptype;
      int               numargs;
      Oid            *argtypes;
      char    **argnames;
      char     *argmodes;
      bool        is_dml_trigger = false;
      bool        is_event_trigger = false;
      int               i;

      /* Get the new function's pg_proc entry */
      tuple = SearchSysCache1(PROCOID, ObjectIdGetDatum(funcoid));
      if (!HeapTupleIsValid(tuple))
            elog(ERROR, "cache lookup failed for function %u", funcoid);
      proc = (Form_pg_proc) GETSTRUCT(tuple);

      functyptype = get_typtype(proc->prorettype);
```

```
      /* Disallow pseudotype result */
      /* except for TRIGGER, RECORD, VOID, or polymorphic */
      if (functyptype == TYPTYPE_PSEUDO)
      {
            /* we assume OPAQUE with no arguments means a trigger */
            if (proc->prorettype == TRIGGEROID ||
                  (proc->prorettype == OPAQUEOID && proc->pronargs == 0))
                  is_dml_trigger = true;
            else if (proc->prorettype == EVTTRIGGEROID)
                  is_event_trigger = true;
            else if (proc->prorettype != RECORDOID &&
                        proc->prorettype != VOIDOID &&
                        !IsPolymorphicType(proc->prorettype))
                  ereport(ERROR,
                              (errcode(ERRCODE_FEATURE_NOT_SUPPORTED),
                               errmsg("PL/pgSQL functions cannot return
type %s",
                                          format_type_be(proc-
>prorettype))));
      }

      /* Disallow pseudotypes in arguments (either IN or OUT) */
      /* except for polymorphic */
      numargs = get_func_arg_info(tuple,
                                                &argtypes, &argnames,
&argmodes);
      for (i = 0; i < numargs; i++)
      {
            if (get_typtype(argtypes[i]) == TYPTYPE_PSEUDO)
            {
                  if (!IsPolymorphicType(argtypes[i]))
                        ereport(ERROR,

      (errcode(ERRCODE_FEATURE_NOT_SUPPORTED),
                                    errmsg("PL/pgSQL functions cannot
accept type %s",

      format_type_be(argtypes[i]))));
            }
      }

      /* Postpone body checks if !check_function_bodies */
      if (check_function_bodies)
      {
            FunctionCallInfoData fake_fcinfo;
            FmgrInfo   flinfo;
            int              rc;
            TriggerData trigdata;
            EventTriggerData etrigdata;

            /*
             * Connect to SPI manager (is this needed for compilation?)
             */
```

```
            if ((rc = SPI_connect()) != SPI_OK_CONNECT)
                  elog(ERROR, "SPI_connect failed: %s",
SPI_result_code_string(rc));

            /*
             * Set up a fake fcinfo with just enough info to satisfy
             * plpgsql_compile().
             */
            MemSet(&fake_fcinfo, 0, sizeof(fake_fcinfo));
            MemSet(&flinfo, 0, sizeof(flinfo));
            fake_fcinfo.flinfo = &flinfo;
            flinfo.fn_oid = funcoid;
            flinfo.fn_mcxt = CurrentMemoryContext;
            if (is_dml_trigger)
            {
                  MemSet(&trigdata, 0, sizeof(trigdata));
                  trigdata.type = T_TriggerData;
                  fake_fcinfo.context = (Node *) &trigdata;
            }
            else if (is_event_trigger)
            {
                  MemSet(&etrigdata, 0, sizeof(etrigdata));
                  etrigdata.type = T_EventTriggerData;
                  fake_fcinfo.context = (Node *) &etrigdata;
            }

            /* Test-compile the function */
            plpgsql_compile(&fake_fcinfo, true);

            /*
             * Disconnect from SPI manager
             */
            if ((rc = SPI_finish()) != SPI_OK_FINISH)
                  elog(ERROR, "SPI_finish failed: %s",
SPI_result_code_string(rc));
      }

      ReleaseSysCache(tuple);

      PG_RETURN_VOID();
}
<sep>
QPDFNameTreeObjectHelper::updateMap(QPDFObjectHandle oh)
{
    if (this->m->seen.count(oh.getObjGen()))
    {
        return;
    }
    this->m->seen.insert(oh.getObjGen());
    QPDFObjectHandle names = oh.getKey("/Names");
    if (names.isArray())
    {
        size_t nitems = names.getArrayNItems();
        size_t i = 0;
```

```
        while (i < nitems - 1)
        {
            QPDFObjectHandle name = names.getArrayItem(i);
            if (name.isString())
            {
                ++i;
                QPDFObjectHandle obj = names.getArrayItem(i);
                this->m->entries[name.getUTF8Value()] = obj;
            }
            ++i;
        }
    }
    QPDFObjectHandle kids = oh.getKey("/Kids");
    if (kids.isArray())
    {
        size_t nitems = kids.getArrayNItems();
        for (size_t i = 0; i < nitems; ++i)
        {
            updateMap(kids.getArrayItem(i));
        }
    }
}
<sep>
static int oidc_handle_discovery_response(request_rec *r, oidc_cfg *c) {

      /* variables to hold the values returned in the response */
      char *issuer = NULL, *target_link_uri = NULL, *login_hint = NULL,
                  *auth_request_params = NULL, *csrf_cookie, *csrf_query =
NULL,
                  *user = NULL, *path_scopes;
      oidc_provider_t *provider = NULL;

      oidc_util_get_request_parameter(r, OIDC_DISC_OP_PARAM, &issuer);
      oidc_util_get_request_parameter(r, OIDC_DISC_USER_PARAM, &user);
      oidc_util_get_request_parameter(r, OIDC_DISC_RT_PARAM,
&target_link_uri);
      oidc_util_get_request_parameter(r, OIDC_DISC_LH_PARAM,
&login_hint);
      oidc_util_get_request_parameter(r, OIDC_DISC_SC_PARAM,
&path_scopes);
      oidc_util_get_request_parameter(r, OIDC_DISC_AR_PARAM,
                  &auth_request_params);
      oidc_util_get_request_parameter(r, OIDC_CSRF_NAME, &csrf_query);
      csrf_cookie = oidc_util_get_cookie(r, OIDC_CSRF_NAME);

      /* do CSRF protection if not 3rd party initiated SSO */
      if (csrf_cookie) {

            /* clean CSRF cookie */
            oidc_util_set_cookie(r, OIDC_CSRF_NAME, "", 0,
                        OIDC_COOKIE_EXT_SAME_SITE_NONE(r));

            /* compare CSRF cookie value with query parameter value */
            if ((csrf_query == NULL)
```

```
                    || apr_strnatcmp(csrf_query, csrf_cookie) != 0) {
                oidc_warn(r,
                          "CSRF protection failed, no Discovery and
dynamic client registration will be allowed");
                csrf_cookie = NULL;
          }
     }

     // TODO: trim issuer/accountname/domain input and do more input
validation

     oidc_debug(r,
                "issuer=\"%s\", target_link_uri=\"%s\",
login_hint=\"%s\", user=\"%s\"",
                issuer, target_link_uri, login_hint, user);

     if (target_link_uri == NULL) {
          if (c->default_sso_url == NULL) {
                return oidc_util_html_send_error(r, c->error_template,
                          "Invalid Request",
                          "SSO to this module without specifying a
\"target_link_uri\" parameter is not possible because " OIDCDefaultURL "
is not set.",
                          HTTP_INTERNAL_SERVER_ERROR);
          }
          target_link_uri = c->default_sso_url;
     }

     /* do open redirect prevention */
     if (oidc_target_link_uri_matches_configuration(r, c,
target_link_uri)
                == FALSE) {
          return oidc_util_html_send_error(r, c->error_template,
                     "Invalid Request",
                     "\"target_link_uri\" parameter does not match
configuration settings, aborting to prevent an open redirect.",
                     HTTP_UNAUTHORIZED);
     }

     /* see if this is a static setup */
     if (c->metadata_dir == NULL) {
          if ((oidc_provider_static_config(r, c, &provider) == TRUE)
                     && (issuer != NULL)) {
                if (apr_strnatcmp(provider->issuer, issuer) != 0) {
                     return oidc_util_html_send_error(r, c-
>error_template,
                                    "Invalid Request",
                                    apr_psprintf(r->pool,
                                              "The \"iss\" value must
match the configured providers' one (%s != %s).",
                                              issuer, c-
>provider.issuer),

     HTTP_INTERNAL_SERVER_ERROR);
```

```
                }
            }
            return oidc_authenticate_user(r, c, NULL, target_link_uri,
login_hint,
                        NULL, NULL, auth_request_params, path_scopes);
      }

      /* find out if the user entered an account name or selected an OP
manually */
      if (user != NULL) {

            if (login_hint == NULL)
                  login_hint = apr_pstrdup(r->pool, user);

            /* normalize the user identifier */
            if (strstr(user, "https://") != user)
                  user = apr_psprintf(r->pool, "https://%s", user);

            /* got an user identifier as input, perform OP discovery with
that */
            if (oidc_proto_url_based_discovery(r, c, user, &issuer) ==
FALSE) {

                  /* something did not work out, show a user facing error
*/
                  return oidc_util_html_send_error(r, c->error_template,
                              "Invalid Request",
                              "Could not resolve the provided user
identifier to an OpenID Connect provider; check your syntax.",
                              HTTP_NOT_FOUND);
            }

            /* issuer is set now, so let's continue as planned */

      } else if (strstr(issuer, OIDC_STR_AT) != NULL) {

            if (login_hint == NULL) {
                  login_hint = apr_pstrdup(r->pool, issuer);
                  //char *p = strstr(issuer, OIDC_STR_AT);
                  //*p = '\0';
            }

            /* got an account name as input, perform OP discovery with
that */
            if (oidc_proto_account_based_discovery(r, c, issuer, &issuer)
                        == FALSE) {

                  /* something did not work out, show a user facing error
*/
                  return oidc_util_html_send_error(r, c->error_template,
                              "Invalid Request",
                              "Could not resolve the provided account name
to an OpenID Connect provider; check your syntax.",
                              HTTP_NOT_FOUND);
```

```
            }

            /* issuer is set now, so let's continue as planned */

      }

      /* strip trailing '/' */
      int n = strlen(issuer);
      if (issuer[n - 1] == OIDC_CHAR_FORWARD_SLASH)
            issuer[n - 1] = '\0';

      /* try and get metadata from the metadata directories for the
selected OP */
      if ((oidc_metadata_get(r, c, issuer, &provider, csrf_cookie !=
NULL) == TRUE)
                  && (provider != NULL)) {

            /* now we've got a selected OP, send the user there to
authenticate */
            return oidc_authenticate_user(r, c, provider,
target_link_uri,
                        login_hint, NULL, NULL, auth_request_params,
path_scopes);
      }

      /* something went wrong */
      return oidc_util_html_send_error(r, c->error_template, "Invalid
Request",
                  "Could not find valid provider metadata for the selected
OpenID Connect provider; contact the administrator",
                  HTTP_NOT_FOUND);
}
<sep>
void vnc_tight_clear(VncState *vs)
{
    int i;
    for (i=0; i<ARRAY_SIZE(vs->tight.stream); i++) {
        if (vs->tight.stream[i].opaque) {
            deflateEnd(&vs->tight.stream[i]);
        }
    }

    buffer_free(&vs->tight.tight);
    buffer_free(&vs->tight.zlib);
    buffer_free(&vs->tight.gradient);
#ifdef CONFIG_VNC_JPEG
    buffer_free(&vs->tight.jpeg);
#endif
#ifdef CONFIG_VNC_PNG
    buffer_free(&vs->tight.png);
#endif
}
<sep>
void ipc_rcu_getref(void *ptr)
```

```
{
      container_of(ptr, struct ipc_rcu_hdr, data)->refcount++;
}
<sep>
void ConnectionManagerImpl::ActiveStream::decodeHeaders(HeaderMapPtr&&
headers, bool end_stream) {
  ScopeTrackerScopeState scope(this,
                               connection_manager_.read_callbacks_-
>connection().dispatcher());
  request_headers_ = std::move(headers);
  // For Admin thread, we don't use routeConfigProvider or SRDS route
provider.
  if (dynamic_cast<Server::Admin*>(&connection_manager_.config_) ==
nullptr &&
      connection_manager_.config_.scopedRouteConfigProvider() != nullptr)
{
    ASSERT(snapped_route_config_ == nullptr,
           "Route config already latched to the active stream when scoped
RDS is enabled.");
    // We need to snap snapped_route_config_ here as it's used in
mutateRequestHeaders later.
    snapScopedRouteConfig();
  }

  if (Http::Headers::get().MethodValues.Head ==
      request_headers_->Method()->value().getStringView()) {
    is_head_request_ = true;
  }
  ENVOY_STREAM_LOG(debug, "request headers complete
(end_stream={}):\n{}", *this, end_stream,
                   *request_headers_);

  // We end the decode here only if the request is header only. If we
convert the request to a
  // header only, the stream will be marked as done once a subsequent
decodeData/decodeTrailers is
  // called with end_stream=true.
  maybeEndDecode(end_stream);

  // Drop new requests when overloaded as soon as we have decoded the
headers.
  if (connection_manager_.overload_stop_accepting_requests_ref_ ==
      Server::OverloadActionState::Active) {
    // In this one special case, do not create the filter chain. If there
is a risk of memory
    // overload it is more important to avoid unnecessary allocation than
to create the filters.
    state_.created_filter_chain_ = true;

connection_manager_.stats_.named_.downstream_rq_overload_close_.inc();
    sendLocalReply(Grpc::Common::hasGrpcContentType(*request_headers_),
                   Http::Code::ServiceUnavailable, "envoy overloaded",
nullptr, is_head_request_,
```

```
                    absl::nullopt,
StreamInfo::ResponseCodeDetails::get().Overload);
    return;
  }

  if (!connection_manager_.config_.proxy100Continue() &&
request_headers_->Expect() &&
      request_headers_->Expect()->value() ==
Headers::get().ExpectValues._100Continue.c_str()) {
    // Note in the case Envoy is handling 100-Continue complexity, it
skips the filter chain
    // and sends the 100-Continue directly to the encoder.
    chargeStats(continueHeader());
    response_encoder_->encode100ContinueHeaders(continueHeader());
    // Remove the Expect header so it won't be handled again upstream.
    request_headers_->removeExpect();
  }

  connection_manager_.user_agent_.initializeFromHeaders(
      *request_headers_, connection_manager_.stats_.prefix_,
connection_manager_.stats_.scope_);

  // Make sure we are getting a codec version we support.
  Protocol protocol = connection_manager_.codec_->protocol();
  if (protocol == Protocol::Http10) {
    // Assume this is HTTP/1.0. This is fine for HTTP/0.9 but this code
will also affect any
    // requests with non-standard version numbers (0.9, 1.3), basically
anything which is not
    // HTTP/1.1.
    //
    // The protocol may have shifted in the HTTP/1.0 case so reset it.
    stream_info_.protocol(protocol);
    if (!connection_manager_.config_.http1Settings().accept_http_10_) {
      // Send "Upgrade Required" if HTTP/1.0 support is not explicitly
configured on.
      sendLocalReply(false, Code::UpgradeRequired, "", nullptr,
is_head_request_, absl::nullopt,
                     StreamInfo::ResponseCodeDetails::get().LowVersion);
      return;
    } else {
      // HTTP/1.0 defaults to single-use connections. Make sure the
connection
      // will be closed unless Keep-Alive is present.
      state_.saw_connection_close_ = true;
      if (request_headers_->Connection() &&
          absl::EqualsIgnoreCase(request_headers_->Connection()-
>value().getStringView(),

Http::Headers::get().ConnectionValues.KeepAlive)) {
        state_.saw_connection_close_ = false;
      }
    }
  }
```

```
  if (!request_headers_->Host()) {
    if ((protocol == Protocol::Http10) &&

!connection_manager_.config_.http1Settings().default_host_for_http_10_.em
pty()) {
      // Add a default host if configured to do so.
      request_headers_->insertHost().value(

connection_manager_.config_.http1Settings().default_host_for_http_10_);
    } else {
      // Require host header. For HTTP/1.1 Host has already been
translated to :authority.
      sendLocalReply(Grpc::Common::hasGrpcContentType(*request_headers_),
Code::BadRequest, "",
                     nullptr, is_head_request_, absl::nullopt,
                     StreamInfo::ResponseCodeDetails::get().MissingHost);
      return;
    }
  }

  ASSERT(connection_manager_.config_.maxRequestHeadersKb() > 0);
  if (request_headers_->byteSize() >
(connection_manager_.config_.maxRequestHeadersKb() * 1024)) {
    sendLocalReply(Grpc::Common::hasGrpcContentType(*request_headers_),
                   Code::RequestHeaderFieldsTooLarge, "", nullptr,
is_head_request_, absl::nullopt,

StreamInfo::ResponseCodeDetails::get().RequestHeadersTooLarge);
    return;
  }

  // Currently we only support relative paths at the application layer.
We expect the codec to have
  // broken the path into pieces if applicable. NOTE: Currently the
HTTP/1.1 codec only does this
  // when the allow_absolute_url flag is enabled on the HCM.
  // https://tools.ietf.org/html/rfc7230#section-5.3 We also need to
check for the existence of
  // :path because CONNECT does not have a path, and we don't support
that currently.
  if (!request_headers_->Path() || request_headers_->Path()-
>value().getStringView().empty() ||
      request_headers_->Path()->value().getStringView()[0] != '/') {
    const bool has_path =
        request_headers_->Path() && !request_headers_->Path()-
>value().getStringView().empty();

connection_manager_.stats_.named_.downstream_rq_non_relative_path_.inc();
    sendLocalReply(Grpc::Common::hasGrpcContentType(*request_headers_),
Code::NotFound, "", nullptr,
                   is_head_request_, absl::nullopt,
                   has_path ?
StreamInfo::ResponseCodeDetails::get().AbsolutePath
```

```
                                :
StreamInfo::ResponseCodeDetails::get().MissingPath);
    return;
  }

  // Path sanitization should happen before any path access other than
the above sanity check.
  if (!ConnectionManagerUtility::maybeNormalizePath(*request_headers_,

connection_manager_.config_)) {
    sendLocalReply(Grpc::Common::hasGrpcContentType(*request_headers_),
Code::BadRequest, "",
                   nullptr, is_head_request_, absl::nullopt,

StreamInfo::ResponseCodeDetails::get().PathNormalizationFailed);
    return;
  }

  if (protocol == Protocol::Http11 && request_headers_->Connection() &&
      absl::EqualsIgnoreCase(request_headers_->Connection()-
>value().getStringView(),

Http::Headers::get().ConnectionValues.Close)) {
    state_.saw_connection_close_ = true;
  }
  // Note: Proxy-Connection is not a standard header, but is supported
here
  // since it is supported by http-parser the underlying parser for http
  // requests.
  if (protocol != Protocol::Http2 && !state_.saw_connection_close_ &&
      request_headers_->ProxyConnection() &&
      absl::EqualsIgnoreCase(request_headers_->ProxyConnection()-
>value().getStringView(),

Http::Headers::get().ConnectionValues.Close)) {
    state_.saw_connection_close_ = true;
  }

  if (!state_.is_internally_created_) { // Only sanitize headers on first
pass.
    // Modify the downstream remote address depending on configuration
and headers.

stream_info_.setDownstreamRemoteAddress(ConnectionManagerUtility::mutateR
equestHeaders(
        *request_headers_, connection_manager_.read_callbacks_-
>connection(),
        connection_manager_.config_, *snapped_route_config_,
connection_manager_.random_generator_,
        connection_manager_.local_info_));
  }
  ASSERT(stream_info_.downstreamRemoteAddress() != nullptr);

  ASSERT(!cached_route_);
```

```
  refreshCachedRoute();

  if (!state_.is_internally_created_) { // Only mutate tracing headers on
first pass.
    ConnectionManagerUtility::mutateTracingRequestHeader(
        *request_headers_, connection_manager_.runtime_,
connection_manager_.config_,
        cached_route_.value().get());
  }

  const bool upgrade_rejected = createFilterChain() == false;

  // TODO if there are no filters when starting a filter iteration, the
connection manager
  // should return 404. The current returns no response if there is no
router filter.
  if (protocol == Protocol::Http11 && hasCachedRoute()) {
    if (upgrade_rejected) {
      // Do not allow upgrades if the route does not support it.

connection_manager_.stats_.named_.downstream_rq_ws_on_non_ws_route_.inc()
;
      sendLocalReply(Grpc::Common::hasGrpcContentType(*request_headers_),
Code::Forbidden, "",
                     nullptr, is_head_request_, absl::nullopt,

StreamInfo::ResponseCodeDetails::get().UpgradeFailed);
      return;
    }
    // Allow non websocket requests to go through websocket enabled
routes.
  }

  if (hasCachedRoute()) {
    const Router::RouteEntry* route_entry = cached_route_.value()-
>routeEntry();
    if (route_entry != nullptr && route_entry->idleTimeout()) {
      idle_timeout_ms_ = route_entry->idleTimeout().value();
      if (idle_timeout_ms_.count()) {
        // If we have a route-level idle timeout but no global stream
idle timeout, create a timer.
        if (stream_idle_timer_ == nullptr) {
          stream_idle_timer_ =
              connection_manager_.read_callbacks_-
>connection().dispatcher().createTimer(
                  [this]() -> void { onIdleTimeout(); });
        }
      } else if (stream_idle_timer_ != nullptr) {
        // If we had a global stream idle timeout but the route-level
idle timeout is set to zero
        // (to override), we disable the idle timer.
        stream_idle_timer_->disableTimer();
        stream_idle_timer_ = nullptr;
      }
```

```
    }
  }

  // Check if tracing is enabled at all.
  if (connection_manager_.config_.tracingConfig()) {
    traceRequest();
  }

  decodeHeaders(nullptr, *request_headers_, end_stream);

  // Reset it here for both global and overridden cases.
  resetIdleTimer();
}
<sep>
static void process_bin_complete_sasl_auth(conn *c) {
    assert(settings.sasl);
    const char *out = NULL;
    unsigned int outlen = 0;

    assert(c->item);
    init_sasl_conn(c);

    int nkey = c->binary_header.request.keylen;
    int vlen = c->binary_header.request.bodylen - nkey;

    char mech[nkey+1];
    memcpy(mech, ITEM_key((item*)c->item), nkey);
    mech[nkey] = 0x00;

    if (settings.verbose)
        fprintf(stderr, "mech:  ``%s'' with %d bytes of data\n", mech,
vlen);

    const char *challenge = vlen == 0 ? NULL : ITEM_data((item*) c-
>item);

    int result=-1;

    switch (c->cmd) {
    case PROTOCOL_BINARY_CMD_SASL_AUTH:
        result = sasl_server_start(c->sasl_conn, mech,
                                   challenge, vlen,
                                   &out, &outlen);
        break;
    case PROTOCOL_BINARY_CMD_SASL_STEP:
        result = sasl_server_step(c->sasl_conn,
                                  challenge, vlen,
                                  &out, &outlen);
        break;
    default:
        assert(false); /* CMD should be one of the above */
        /* This code is pretty much impossible, but makes the compiler
           happier */
        if (settings.verbose) {
```

```
            fprintf(stderr, "Unhandled command %d with challenge %s\n",
                    c->cmd, challenge);
        }
        break;
    }

    item_unlink(c->item);

    if (settings.verbose) {
        fprintf(stderr, "sasl result code:  %d\n", result);
    }

    switch(result) {
    case SASL_OK:
        write_bin_response(c, "Authenticated", 0, 0,
strlen("Authenticated"));
        pthread_mutex_lock(&c->thread->stats.mutex);
        c->thread->stats.auth_cmds++;
        pthread_mutex_unlock(&c->thread->stats.mutex);
        break;
    case SASL_CONTINUE:
        add_bin_header(c, PROTOCOL_BINARY_RESPONSE_AUTH_CONTINUE, 0, 0,
outlen);
        if(outlen > 0) {
            add_iov(c, out, outlen);
        }
        conn_set_state(c, conn_mwrite);
        c->write_and_go = conn_new_cmd;
        break;
    default:
        if (settings.verbose)
            fprintf(stderr, "Unknown sasl response:  %d\n", result);
        write_bin_error(c, PROTOCOL_BINARY_RESPONSE_AUTH_ERROR, 0);
        pthread_mutex_lock(&c->thread->stats.mutex);
        c->thread->stats.auth_cmds++;
        c->thread->stats.auth_errors++;
        pthread_mutex_unlock(&c->thread->stats.mutex);
    }
}
<sep>
static __inline__ int scm_send(struct socket *sock, struct msghdr *msg,
                       struct scm_cookie *scm)
{
      scm_set_cred(scm, task_tgid(current), current_cred());
      scm->fp = NULL;
      unix_get_peersec_dgram(sock, scm);
      if (msg->msg_controllen <= 0)
            return 0;
      return __scm_send(sock, msg, scm);
}
<sep>
static void php_build_argv(char *s, zval *track_vars_array TSRMLS_DC)
{
      zval *arr, *argc, *tmp;
```

```
      int count = 0;
      char *ss, *space;

      if (!(SG(request_info).argc || track_vars_array)) {
            return;
      }

      ALLOC_INIT_ZVAL(arr);
      array_init(arr);

      /* Prepare argv */
      if (SG(request_info).argc) { /* are we in cli sapi? */
            int i;
            for (i = 0; i < SG(request_info).argc; i++) {
                  ALLOC_ZVAL(tmp);
                  Z_TYPE_P(tmp) = IS_STRING;
                  Z_STRLEN_P(tmp) = strlen(SG(request_info).argv[i]);
                  Z_STRVAL_P(tmp) = estrndup(SG(request_info).argv[i],
Z_STRLEN_P(tmp));
                  INIT_PZVAL(tmp);
                  if (zend_hash_next_index_insert(Z_ARRVAL_P(arr), &tmp,
sizeof(zval *), NULL) == FAILURE) {
                        if (Z_TYPE_P(tmp) == IS_STRING) {
                              efree(Z_STRVAL_P(tmp));
                        }
                  }
            }
      } else      if (s && *s) {
            ss = s;
            while (ss) {
                  space = strchr(ss, '+');
                  if (space) {
                        *space = '\0';
                  }
                  /* auto-type */
                  ALLOC_ZVAL(tmp);
                  Z_TYPE_P(tmp) = IS_STRING;
                  Z_STRLEN_P(tmp) = strlen(ss);
                  Z_STRVAL_P(tmp) = estrndup(ss, Z_STRLEN_P(tmp));
                  INIT_PZVAL(tmp);
                  count++;
                  if (zend_hash_next_index_insert(Z_ARRVAL_P(arr), &tmp,
sizeof(zval *), NULL) == FAILURE) {
                        if (Z_TYPE_P(tmp) == IS_STRING) {
                              efree(Z_STRVAL_P(tmp));
                        }
                  }
                  if (space) {
                        *space = '+';
                        ss = space + 1;
                  } else {
                        ss = space;
                  }
            }
```

```
	}

	/* prepare argc */
	ALLOC_INIT_ZVAL(argc);
	if (SG(request_info).argc) {
		Z_LVAL_P(argc) = SG(request_info).argc;
	} else {
		Z_LVAL_P(argc) = count;
	}
	Z_TYPE_P(argc) = IS_LONG;

	if (SG(request_info).argc) {
		Z_ADDREF_P(arr);
		Z_ADDREF_P(argc);
		zend_hash_update(&EG(symbol_table), "argv", sizeof("argv"),
&arr, sizeof(zval *), NULL);
		zend_hash_update(&EG(symbol_table), "argc", sizeof("argc"),
&argc, sizeof(zval *), NULL);
	}
	if (track_vars_array) {
		Z_ADDREF_P(arr);
		Z_ADDREF_P(argc);
		zend_hash_update(Z_ARRVAL_P(track_vars_array), "argv",
sizeof("argv"), &arr, sizeof(zval *), NULL);
		zend_hash_update(Z_ARRVAL_P(track_vars_array), "argc",
sizeof("argc"), &argc, sizeof(zval *), NULL);
	}
	zval_ptr_dtor(&arr);
	zval_ptr_dtor(&argc);
}
<sep>
BGD_DECLARE(gdImagePtr) gdImageCropThreshold(gdImagePtr im, const
unsigned int color, const float threshold)
{
	const int width = gdImageSX(im);
	const int height = gdImageSY(im);

	int x,y;
	int match;
	gdRect crop;

	crop.x = 0;
	crop.y = 0;
	crop.width = 0;
	crop.height = 0;

	/* Pierre: crop everything sounds bad */
	if (threshold > 100.0) {
		return NULL;
	}

	/* TODO: Add gdImageGetRowPtr and works with ptr at the row level
	 * for the true color and palette images
	 * new formats will simply work with ptr
```

```
       */
      match = 1;
      for (y = 0; match && y < height; y++) {
            for (x = 0; match && x < width; x++) {
                  match = (gdColorMatch(im, color, gdImageGetPixel(im,
x,y), threshold)) > 0;
            }
      }

      /* Pierre
       * Nothing to do > bye
       * Duplicate the image?
       */
      if (y == height - 1) {
            return NULL;
      }

      crop.y = y -1;
      match = 1;
      for (y = height - 1; match && y >= 0; y--) {
            for (x = 0; match && x < width; x++) {
                  match = (gdColorMatch(im, color, gdImageGetPixel(im, x,
y), threshold)) > 0;
            }
      }

      if (y == 0) {
            crop.height = height - crop.y + 1;
      } else {
            crop.height = y - crop.y + 2;
      }

      match = 1;
      for (x = 0; match && x < width; x++) {
            for (y = 0; match && y < crop.y + crop.height - 1; y++) {
                  match = (gdColorMatch(im, color, gdImageGetPixel(im,
x,y), threshold)) > 0;
            }
      }
      crop.x = x - 1;

      match = 1;
      for (x = width - 1; match && x >= 0; x--) {
            for (y = 0; match &&  y < crop.y + crop.height - 1; y++) {
                  match = (gdColorMatch(im, color, gdImageGetPixel(im,
x,y), threshold)) > 0;
            }
      }
      crop.width = x - crop.x + 2;

      return gdImageCrop(im, &crop);
}
<sep>
static int cloop_open(BlockDriverState *bs, QDict *options, int flags,
```

```
                    Error **errp)
{
    BDRVCloopState *s = bs->opaque;
    uint32_t offsets_size, max_compressed_block_size = 1, i;
    int ret;

    bs->read_only = 1;

    /* read header */
    ret = bdrv_pread(bs->file, 128, &s->block_size, 4);
    if (ret < 0) {
        return ret;
    }
    s->block_size = be32_to_cpu(s->block_size);

    ret = bdrv_pread(bs->file, 128 + 4, &s->n_blocks, 4);
    if (ret < 0) {
        return ret;
    }
    s->n_blocks = be32_to_cpu(s->n_blocks);

    /* read offsets */
    offsets_size = s->n_blocks * sizeof(uint64_t);
    s->offsets = g_malloc(offsets_size);

    ret = bdrv_pread(bs->file, 128 + 4 + 4, s->offsets, offsets_size);
    if (ret < 0) {
        goto fail;
    }

    for(i=0;i<s->n_blocks;i++) {
        s->offsets[i] = be64_to_cpu(s->offsets[i]);
        if (i > 0) {
            uint32_t size = s->offsets[i] - s->offsets[i - 1];
            if (size > max_compressed_block_size) {
                max_compressed_block_size = size;
            }
        }
    }

    /* initialize zlib engine */
    s->compressed_block = g_malloc(max_compressed_block_size + 1);
    s->uncompressed_block = g_malloc(s->block_size);
    if (inflateInit(&s->zstream) != Z_OK) {
        ret = -EINVAL;
        goto fail;
    }
    s->current_block = s->n_blocks;

    s->sectors_per_block = s->block_size/512;
    bs->total_sectors = s->n_blocks * s->sectors_per_block;
    qemu_co_mutex_init(&s->lock);
    return 0;
```

```
fail:
    g_free(s->offsets);
    g_free(s->compressed_block);
    g_free(s->uncompressed_block);
    return ret;
}
<sep>
static int zr364xx_vidioc_querycap(struct file *file, void *priv,
                          struct v4l2_capability *cap)
{
      struct zr364xx_camera *cam = video_drvdata(file);

      strscpy(cap->driver, DRIVER_DESC, sizeof(cap->driver));
      strscpy(cap->card, cam->udev->product, sizeof(cap->card));
      strscpy(cap->bus_info, dev_name(&cam->udev->dev),
            sizeof(cap->bus_info));
      cap->device_caps = V4L2_CAP_VIDEO_CAPTURE |
                     V4L2_CAP_READWRITE |
                     V4L2_CAP_STREAMING;
      cap->capabilities = cap->device_caps | V4L2_CAP_DEVICE_CAPS;

      return 0;
}
<sep>
static int manager_dispatch_notify_fd(sd_event_source *source, int fd,
uint32_t revents, void *userdata) {

        _cleanup_fdset_free_ FDSet *fds = NULL;
        Manager *m = userdata;
        char buf[NOTIFY_BUFFER_MAX+1];
        struct iovec iovec = {
                .iov_base = buf,
                .iov_len = sizeof(buf)-1,
        };
        union {
                struct cmsghdr cmsghdr;
                uint8_t buf[CMSG_SPACE(sizeof(struct ucred)) +
                            CMSG_SPACE(sizeof(int) * NOTIFY_FD_MAX)];
        } control = {};
        struct msghdr msghdr = {
                .msg_iov = &iovec,
                .msg_iovlen = 1,
                .msg_control = &control,
                .msg_controllen = sizeof(control),
        };

        struct cmsghdr *cmsg;
        struct ucred *ucred = NULL;
        bool found = false;
        Unit *u1, *u2, *u3;
        int r, *fd_array = NULL;
        unsigned n_fds = 0;
        ssize_t n;
```

```
        assert(m);
        assert(m->notify_fd == fd);

        if (revents != EPOLLIN) {
                log_warning("Got unexpected poll event for notify fd.");
                return 0;
        }

        n = recvmsg(m->notify_fd, &msghdr,
MSG_DONTWAIT|MSG_CMSG_CLOEXEC);
        if (n < 0) {
                if (errno == EAGAIN || errno == EINTR)
                        return 0;

                return -errno;
        }
        if (n == 0) {
                log_debug("Got zero-length notification message.
Ignoring.");
                return 0;
        }

        CMSG_FOREACH(cmsg, &msghdr) {
                if (cmsg->cmsg_level == SOL_SOCKET && cmsg->cmsg_type ==
SCM_RIGHTS) {

                        fd_array = (int*) CMSG_DATA(cmsg);
                        n_fds = (cmsg->cmsg_len - CMSG_LEN(0)) /
sizeof(int);

                } else if (cmsg->cmsg_level == SOL_SOCKET &&
                           cmsg->cmsg_type == SCM_CREDENTIALS &&
                           cmsg->cmsg_len == CMSG_LEN(sizeof(struct
ucred))) {

                        ucred = (struct ucred*) CMSG_DATA(cmsg);
                }
        }

        if (n_fds > 0) {
                assert(fd_array);

                r = fdset_new_array(&fds, fd_array, n_fds);
                if (r < 0) {
                        close_many(fd_array, n_fds);
                        return log_oom();
                }
        }

        if (!ucred || ucred->pid <= 0) {
                log_warning("Received notify message without valid
credentials. Ignoring.");
                return 0;
        }
```

```
        if ((size_t) n >= sizeof(buf)) {
                log_warning("Received notify message exceeded maximum
size. Ignoring.");
                return 0;
        }

        buf[n] = 0;

        /* Notify every unit that might be interested, but try
         * to avoid notifying the same one multiple times. */
        u1 = manager_get_unit_by_pid_cgroup(m, ucred->pid);
        if (u1) {
                manager_invoke_notify_message(m, u1, ucred->pid, buf, n,
fds);
                found = true;
        }

        u2 = hashmap_get(m->watch_pids1, PID_TO_PTR(ucred->pid));
        if (u2 && u2 != u1) {
                manager_invoke_notify_message(m, u2, ucred->pid, buf, n,
fds);
                found = true;
        }

        u3 = hashmap_get(m->watch_pids2, PID_TO_PTR(ucred->pid));
        if (u3 && u3 != u2 && u3 != u1) {
                manager_invoke_notify_message(m, u3, ucred->pid, buf, n,
fds);
                found = true;
        }

        if (!found)
                log_warning("Cannot find unit for notify message of PID
"PID_FMT".", ucred->pid);

        if (fdset_size(fds) > 0)
                log_warning("Got auxiliary fds with notification message,
closing all.");

        return 0;
}
<sep>
static void sas_probe_devices(struct work_struct *work)
{
      struct domain_device *dev, *n;
      struct sas_discovery_event *ev = to_sas_discovery_event(work);
      struct asd_sas_port *port = ev->port;

      clear_bit(DISCE_PROBE, &port->disc.pending);

      /* devices must be domain members before link recovery and probe */
      list_for_each_entry(dev, &port->disco_list, disco_list_node) {
            spin_lock_irq(&port->dev_list_lock);
```

```
            list_add_tail(&dev->dev_list_node, &port->dev_list);
            spin_unlock_irq(&port->dev_list_lock);
      }

      sas_probe_sata(port);

      list_for_each_entry_safe(dev, n, &port->disco_list,
disco_list_node) {
            int err;

            err = sas_rphy_add(dev->rphy);
            if (err)
                  sas_fail_probe(dev, __func__, err);
            else
                  list_del_init(&dev->disco_list_node);
      }
}
<sep>
static Image *ReadVIFFImage(const ImageInfo *image_info,
  ExceptionInfo *exception)
{
#define VFF_CM_genericRGB  15
#define VFF_CM_ntscRGB  1
#define VFF_CM_NONE  0
#define VFF_DEP_DECORDER  0x4
#define VFF_DEP_NSORDER  0x8
#define VFF_DES_RAW  0
#define VFF_LOC_IMPLICIT  1
#define VFF_MAPTYP_NONE  0
#define VFF_MAPTYP_1_BYTE  1
#define VFF_MAPTYP_2_BYTE  2
#define VFF_MAPTYP_4_BYTE  4
#define VFF_MAPTYP_FLOAT  5
#define VFF_MAPTYP_DOUBLE  7
#define VFF_MS_NONE  0
#define VFF_MS_ONEPERBAND  1
#define VFF_MS_SHARED  3
#define VFF_TYP_BIT  0
#define VFF_TYP_1_BYTE  1
#define VFF_TYP_2_BYTE  2
#define VFF_TYP_4_BYTE  4
#define VFF_TYP_FLOAT  5
#define VFF_TYP_DOUBLE  9

  typedef struct _ViffInfo
  {
    unsigned char
      identifier,
      file_type,
      release,
      version,
      machine_dependency,
      reserve[3];
```

```
  char
    comment[512];

  unsigned int
    rows,
    columns,
    subrows;

  int
    x_offset,
    y_offset;

  float
    x_bits_per_pixel,
    y_bits_per_pixel;

  unsigned int
    location_type,
    location_dimension,
    number_of_images,
    number_data_bands,
    data_storage_type,
    data_encode_scheme,
    map_scheme,
    map_storage_type,
    map_rows,
    map_columns,
    map_subrows,
    map_enable,
    maps_per_cycle,
    color_space_model;
} ViffInfo;

double
  min_value,
  scale_factor,
  value;

Image
  *image;

int
  bit;

MagickBooleanType
  status;

MagickSizeType
  number_pixels;

register ssize_t
  x;

register Quantum
```

```
  *q;

register ssize_t
  i;

register unsigned char
  *p;

size_t
  bytes_per_pixel,
  max_packets,
  quantum;

ssize_t
  count,
  y;

unsigned char
  *pixels;

unsigned long
  lsb_first;

ViffInfo
  viff_info;

/*
  Open image file.
*/
assert(image_info != (const ImageInfo *) NULL);
assert(image_info->signature == MagickCoreSignature);
if (image_info->debug != MagickFalse)
  (void) LogMagickEvent(TraceEvent,GetMagickModule(),"%s",
    image_info->filename);
assert(exception != (ExceptionInfo *) NULL);
assert(exception->signature == MagickCoreSignature);
image=AcquireImage(image_info,exception);
status=OpenBlob(image_info,image,ReadBinaryBlobMode,exception);
if (status == MagickFalse)
  {
    image=DestroyImageList(image);
    return((Image *) NULL);
  }
/*
  Read VIFF header (1024 bytes).
*/
count=ReadBlob(image,1,&viff_info.identifier);
do
{
  /*
    Verify VIFF identifier.
  */
  if ((count != 1) || ((unsigned char) viff_info.identifier != 0xab))
    ThrowReaderException(CorruptImageError,"NotAVIFFImage");
```

```
    /*
      Initialize VIFF image.
    */
    (void)
ReadBlob(image,sizeof(viff_info.file_type),&viff_info.file_type);
    (void) ReadBlob(image,sizeof(viff_info.release),&viff_info.release);
    (void) ReadBlob(image,sizeof(viff_info.version),&viff_info.version);
    (void) ReadBlob(image,sizeof(viff_info.machine_dependency),
      &viff_info.machine_dependency);
    (void) ReadBlob(image,sizeof(viff_info.reserve),viff_info.reserve);
    count=ReadBlob(image,512,(unsigned char *) viff_info.comment);
    viff_info.comment[511]='\0';
    if (strlen(viff_info.comment) > 4)
      (void)
SetImageProperty(image,"comment",viff_info.comment,exception);
    if ((viff_info.machine_dependency == VFF_DEP_DECORDER) ||
        (viff_info.machine_dependency == VFF_DEP_NSORDER))
      image->endian=LSBEndian;
    else
      image->endian=MSBEndian;
    viff_info.rows=ReadBlobLong(image);
    viff_info.columns=ReadBlobLong(image);
    viff_info.subrows=ReadBlobLong(image);
    viff_info.x_offset=(int) ReadBlobLong(image);
    viff_info.y_offset=(int) ReadBlobLong(image);
    viff_info.x_bits_per_pixel=(float) ReadBlobLong(image);
    viff_info.y_bits_per_pixel=(float) ReadBlobLong(image);
    viff_info.location_type=ReadBlobLong(image);
    viff_info.location_dimension=ReadBlobLong(image);
    viff_info.number_of_images=ReadBlobLong(image);
    viff_info.number_data_bands=ReadBlobLong(image);
    viff_info.data_storage_type=ReadBlobLong(image);
    viff_info.data_encode_scheme=ReadBlobLong(image);
    viff_info.map_scheme=ReadBlobLong(image);
    viff_info.map_storage_type=ReadBlobLong(image);
    viff_info.map_rows=ReadBlobLong(image);
    viff_info.map_columns=ReadBlobLong(image);
    viff_info.map_subrows=ReadBlobLong(image);
    viff_info.map_enable=ReadBlobLong(image);
    viff_info.maps_per_cycle=ReadBlobLong(image);
    viff_info.color_space_model=ReadBlobLong(image);
    for (i=0; i < 420; i++)
      (void) ReadBlobByte(image);
    if (EOFBlob(image) != MagickFalse)
      ThrowReaderException(CorruptImageError,"UnexpectedEndOfFile");
    image->columns=viff_info.rows;
    image->rows=viff_info.columns;
    image->depth=viff_info.x_bits_per_pixel <= 8 ? 8UL :
      MAGICKCORE_QUANTUM_DEPTH;
    /*
      Verify that we can read this VIFF image.
    */
    number_pixels=(MagickSizeType) viff_info.columns*viff_info.rows;
    if (number_pixels != (size_t) number_pixels)
```

```
      ThrowReaderException(ResourceLimitError,"MemoryAllocationFailed");
    if (number_pixels == 0)

ThrowReaderException(CoderError,"ImageColumnOrRowSizeIsNotSupported");
    if ((viff_info.number_data_bands < 1) || (viff_info.number_data_bands
> 4))
      ThrowReaderException(CorruptImageError,"ImproperImageHeader");
    if ((viff_info.data_storage_type != VFF_TYP_BIT) &&
        (viff_info.data_storage_type != VFF_TYP_1_BYTE) &&
        (viff_info.data_storage_type != VFF_TYP_2_BYTE) &&
        (viff_info.data_storage_type != VFF_TYP_4_BYTE) &&
        (viff_info.data_storage_type != VFF_TYP_FLOAT) &&
        (viff_info.data_storage_type != VFF_TYP_DOUBLE))
      ThrowReaderException(CoderError,"DataStorageTypeIsNotSupported");
    if (viff_info.data_encode_scheme != VFF_DES_RAW)

ThrowReaderException(CoderError,"DataEncodingSchemeIsNotSupported");
    if ((viff_info.map_storage_type != VFF_MAPTYP_NONE) &&
        (viff_info.map_storage_type != VFF_MAPTYP_1_BYTE) &&
        (viff_info.map_storage_type != VFF_MAPTYP_2_BYTE) &&
        (viff_info.map_storage_type != VFF_MAPTYP_4_BYTE) &&
        (viff_info.map_storage_type != VFF_MAPTYP_FLOAT) &&
        (viff_info.map_storage_type != VFF_MAPTYP_DOUBLE))
      ThrowReaderException(CoderError,"MapStorageTypeIsNotSupported");
    if ((viff_info.color_space_model != VFF_CM_NONE) &&
        (viff_info.color_space_model != VFF_CM_ntscRGB) &&
        (viff_info.color_space_model != VFF_CM_genericRGB))
      ThrowReaderException(CoderError,"ColorspaceModelIsNotSupported");
    if (viff_info.location_type != VFF_LOC_IMPLICIT)
      ThrowReaderException(CoderError,"LocationTypeIsNotSupported");
    if (viff_info.number_of_images != 1)
      ThrowReaderException(CoderError,"NumberOfImagesIsNotSupported");
    if (viff_info.map_rows == 0)
      viff_info.map_scheme=VFF_MS_NONE;
    switch ((int) viff_info.map_scheme)
    {
      case VFF_MS_NONE:
      {
        if (viff_info.number_data_bands < 3)
          {
            /*
              Create linear color ramp.
            */
            if (viff_info.data_storage_type == VFF_TYP_BIT)
              image->colors=2;
            else
              if (viff_info.data_storage_type == VFF_MAPTYP_1_BYTE)
                image->colors=256UL;
              else
                image->colors=image->depth <= 8 ? 256UL : 65536UL;
            status=AcquireImageColormap(image,image->colors,exception);
            if (status == MagickFalse)

ThrowReaderException(ResourceLimitError,"MemoryAllocationFailed");
```

```
          }
        break;
      }
      case VFF_MS_ONEPERBAND:
      case VFF_MS_SHARED:
      {
        unsigned char
          *viff_colormap;

        /*
          Allocate VIFF colormap.
        */
        switch ((int) viff_info.map_storage_type)
        {
          case VFF_MAPTYP_1_BYTE: bytes_per_pixel=1; break;
          case VFF_MAPTYP_2_BYTE: bytes_per_pixel=2; break;
          case VFF_MAPTYP_4_BYTE: bytes_per_pixel=4; break;
          case VFF_MAPTYP_FLOAT: bytes_per_pixel=4; break;
          case VFF_MAPTYP_DOUBLE: bytes_per_pixel=8; break;
          default: bytes_per_pixel=1; break;
        }
        image->colors=viff_info.map_columns;
        if (AcquireImageColormap(image,image->colors,exception) ==
MagickFalse)

ThrowReaderException(ResourceLimitError,"MemoryAllocationFailed");
        if (viff_info.map_rows >
            (viff_info.map_rows*bytes_per_pixel*sizeof(*viff_colormap)))
          ThrowReaderException(CorruptImageError,"ImproperImageHeader");
        viff_colormap=(unsigned char *) AcquireQuantumMemory(image-
>colors,
          viff_info.map_rows*bytes_per_pixel*sizeof(*viff_colormap));
        if (viff_colormap == (unsigned char *) NULL)

ThrowReaderException(ResourceLimitError,"MemoryAllocationFailed");
        /*
          Read VIFF raster colormap.
        */
        count=ReadBlob(image,bytes_per_pixel*image-
>colors*viff_info.map_rows,
          viff_colormap);
        lsb_first=1;
        if (*(char *) &lsb_first &&
            ((viff_info.machine_dependency != VFF_DEP_DECORDER) &&
             (viff_info.machine_dependency != VFF_DEP_NSORDER)))
          switch ((int) viff_info.map_storage_type)
          {
            case VFF_MAPTYP_2_BYTE:
            {
              MSBOrderShort(viff_colormap,(bytes_per_pixel*image->colors*
                viff_info.map_rows));
              break;
            }
            case VFF_MAPTYP_4_BYTE:
```

```
            case VFF_MAPTYP_FLOAT:
            {
              MSBOrderLong(viff_colormap,(bytes_per_pixel*image->colors*
                viff_info.map_rows));
              break;
            }
            default: break;
          }
        for (i=0; i < (ssize_t) (viff_info.map_rows*image->colors); i++)
        {
          switch ((int) viff_info.map_storage_type)
          {
            case VFF_MAPTYP_2_BYTE: value=1.0*((short *)
viff_colormap)[i]; break;
            case VFF_MAPTYP_4_BYTE: value=1.0*((int *) viff_colormap)[i];
break;
            case VFF_MAPTYP_FLOAT: value=((float *) viff_colormap)[i];
break;
            case VFF_MAPTYP_DOUBLE: value=((double *) viff_colormap)[i];
break;
            default: value=1.0*viff_colormap[i]; break;
          }
          if (i < (ssize_t) image->colors)
            {
              image->colormap[i].red=ScaleCharToQuantum((unsigned char)
value);
              image->colormap[i].green=
                ScaleCharToQuantum((unsigned char) value);
              image->colormap[i].blue=ScaleCharToQuantum((unsigned char)
value);
            }
          else
            if (i < (ssize_t) (2*image->colors))
              image->colormap[i % image->colors].green=
                ScaleCharToQuantum((unsigned char) value);
            else
              if (i < (ssize_t) (3*image->colors))
                image->colormap[i % image->colors].blue=
                  ScaleCharToQuantum((unsigned char) value);
        }
        viff_colormap=(unsigned char *)
RelinquishMagickMemory(viff_colormap);
        break;
      }
      default:
        ThrowReaderException(CoderError,"ColormapTypeNotSupported");
    }
    /*
      Initialize image structure.
    */
    image->alpha_trait=viff_info.number_data_bands == 4 ? BlendPixelTrait
:
      UndefinedPixelTrait;
    image->storage_class=(viff_info.number_data_bands < 3 ? PseudoClass :
```

```
      DirectClass);
    image->columns=viff_info.rows;
    image->rows=viff_info.columns;
    if ((image_info->ping != MagickFalse) && (image_info->number_scenes
!= 0))
      if (image->scene >= (image_info->scene+image_info->number_scenes-
1))
        break;
    status=SetImageExtent(image,image->columns,image->rows,exception);
    if (status == MagickFalse)
      return(DestroyImageList(image));
    /*
      Allocate VIFF pixels.
    */
    switch ((int) viff_info.data_storage_type)
    {
      case VFF_TYP_2_BYTE: bytes_per_pixel=2; break;
      case VFF_TYP_4_BYTE: bytes_per_pixel=4; break;
      case VFF_TYP_FLOAT: bytes_per_pixel=4; break;
      case VFF_TYP_DOUBLE: bytes_per_pixel=8; break;
      default: bytes_per_pixel=1; break;
    }
    if (viff_info.data_storage_type == VFF_TYP_BIT)
      max_packets=((image->columns+7UL) >> 3UL)*image->rows;
    else
      max_packets=(size_t) (number_pixels*viff_info.number_data_bands);
    pixels=(unsigned char *)
AcquireQuantumMemory(MagickMax(number_pixels,
      max_packets),bytes_per_pixel*sizeof(*pixels));
    if (pixels == (unsigned char *) NULL)
      ThrowReaderException(ResourceLimitError,"MemoryAllocationFailed");
    count=ReadBlob(image,bytes_per_pixel*max_packets,pixels);
    lsb_first=1;
    if (*(char *) &lsb_first &&
        ((viff_info.machine_dependency != VFF_DEP_DECORDER) &&
         (viff_info.machine_dependency != VFF_DEP_NSORDER)))
      switch ((int) viff_info.data_storage_type)
      {
        case VFF_TYP_2_BYTE:
        {
          MSBOrderShort(pixels,bytes_per_pixel*max_packets);
          break;
        }
        case VFF_TYP_4_BYTE:
        case VFF_TYP_FLOAT:
        {
          MSBOrderLong(pixels,bytes_per_pixel*max_packets);
          break;
        }
        default: break;
      }
    min_value=0.0;
    scale_factor=1.0;
    if ((viff_info.data_storage_type != VFF_TYP_1_BYTE) &&
```

```
    (viff_info.map_scheme == VFF_MS_NONE))
  {
    double
      max_value;

    /*
      Determine scale factor.
    */
    switch ((int) viff_info.data_storage_type)
    {
      case VFF_TYP_2_BYTE: value=1.0*((short *) pixels)[0]; break;
      case VFF_TYP_4_BYTE: value=1.0*((int *) pixels)[0]; break;
      case VFF_TYP_FLOAT: value=((float *) pixels)[0]; break;
      case VFF_TYP_DOUBLE: value=((double *) pixels)[0]; break;
      default: value=1.0*pixels[0]; break;
    }
    max_value=value;
    min_value=value;
    for (i=0; i < (ssize_t) max_packets; i++)
    {
      switch ((int) viff_info.data_storage_type)
      {
        case VFF_TYP_2_BYTE: value=1.0*((short *) pixels)[i]; break;
        case VFF_TYP_4_BYTE: value=1.0*((int *) pixels)[i]; break;
        case VFF_TYP_FLOAT: value=((float *) pixels)[i]; break;
        case VFF_TYP_DOUBLE: value=((double *) pixels)[i]; break;
        default: value=1.0*pixels[i]; break;
      }
      if (value > max_value)
        max_value=value;
      else
        if (value < min_value)
          min_value=value;
    }
    if ((min_value == 0) && (max_value == 0))
      scale_factor=0;
    else
      if (min_value == max_value)
        {
          scale_factor=(double) QuantumRange/min_value;
          min_value=0;
        }
      else
        scale_factor=(double) QuantumRange/(max_value-min_value);
  }
/*
  Convert pixels to Quantum size.
*/
p=(unsigned char *) pixels;
for (i=0; i < (ssize_t) max_packets; i++)
{
  switch ((int) viff_info.data_storage_type)
  {
    case VFF_TYP_2_BYTE: value=1.0*((short *) pixels)[i]; break;
```

```
    case VFF_TYP_4_BYTE: value=1.0*((int *) pixels)[i]; break;
    case VFF_TYP_FLOAT: value=((float *) pixels)[i]; break;
    case VFF_TYP_DOUBLE: value=((double *) pixels)[i]; break;
    default: value=1.0*pixels[i]; break;
  }
  if (viff_info.map_scheme == VFF_MS_NONE)
    {
      value=(value-min_value)*scale_factor;
      if (value > QuantumRange)
        value=QuantumRange;
      else
        if (value < 0)
          value=0;
    }
  *p=(unsigned char) ((Quantum) value);
  p++;
}
/*
  Convert VIFF raster image to pixel packets.
*/
p=(unsigned char *) pixels;
if (viff_info.data_storage_type == VFF_TYP_BIT)
  {
    /*
      Convert bitmap scanline.
    */
    for (y=0; y < (ssize_t) image->rows; y++)
    {
      q=QueueAuthenticPixels(image,0,y,image->columns,1,exception);
      if (q == (Quantum *) NULL)
        break;
      for (x=0; x < (ssize_t) (image->columns-7); x+=8)
      {
        for (bit=0; bit < 8; bit++)
        {
          quantum=(size_t) ((*p) & (0x01 << bit) ? 0 : 1);
          SetPixelRed(image,quantum == 0 ? 0 : QuantumRange,q);
          SetPixelGreen(image,quantum == 0 ? 0 : QuantumRange,q);
          SetPixelBlue(image,quantum == 0 ? 0 : QuantumRange,q);
          if (image->storage_class == PseudoClass)
            SetPixelIndex(image,(Quantum) quantum,q);
          q+=GetPixelChannels(image);
        }
        p++;
      }
      if ((image->columns % 8) != 0)
        {
          for (bit=0; bit < (int) (image->columns % 8); bit++)
          {
            quantum=(size_t) ((*p) & (0x01 << bit) ? 0 : 1);
            SetPixelRed(image,quantum == 0 ? 0 : QuantumRange,q);
            SetPixelGreen(image,quantum == 0 ? 0 : QuantumRange,q);
            SetPixelBlue(image,quantum == 0 ? 0 : QuantumRange,q);
            if (image->storage_class == PseudoClass)
```

```
                SetPixelIndex(image,(Quantum) quantum,q);
              q+=GetPixelChannels(image);
            }
            p++;
          }
        if (SyncAuthenticPixels(image,exception) == MagickFalse)
          break;
        if (image->previous == (Image *) NULL)
          {

status=SetImageProgress(image,LoadImageTag,(MagickOffsetType) y,
              image->rows);
            if (status == MagickFalse)
              break;
          }
      }
    }
  else
    if (image->storage_class == PseudoClass)
      for (y=0; y < (ssize_t) image->rows; y++)
      {
        q=QueueAuthenticPixels(image,0,y,image->columns,1,exception);
        if (q == (Quantum *) NULL)
          break;
        for (x=0; x < (ssize_t) image->columns; x++)
        {
          SetPixelIndex(image,*p++,q);
          q+=GetPixelChannels(image);
        }
        if (SyncAuthenticPixels(image,exception) == MagickFalse)
          break;
        if (image->previous == (Image *) NULL)
          {

status=SetImageProgress(image,LoadImageTag,(MagickOffsetType) y,
              image->rows);
            if (status == MagickFalse)
              break;
          }
      }
    else
      {
        /*
          Convert DirectColor scanline.
        */
        number_pixels=(MagickSizeType) image->columns*image->rows;
        for (y=0; y < (ssize_t) image->rows; y++)
        {
          q=QueueAuthenticPixels(image,0,y,image->columns,1,exception);
          if (q == (Quantum *) NULL)
            break;
          for (x=0; x < (ssize_t) image->columns; x++)
          {
            SetPixelRed(image,ScaleCharToQuantum(*p),q);
```

```
SetPixelGreen(image,ScaleCharToQuantum(*(p+number_pixels)),q);

SetPixelBlue(image,ScaleCharToQuantum(*(p+2*number_pixels)),q);
              if (image->colors != 0)
                {
                  ssize_t
                    index;

                  index=(ssize_t) GetPixelRed(image,q);
                  SetPixelRed(image,image->colormap[

ConstrainColormapIndex(image,index,exception)].red,q);
                  index=(ssize_t) GetPixelGreen(image,q);
                  SetPixelGreen(image,image->colormap[

ConstrainColormapIndex(image,index,exception)].green,q);
                  index=(ssize_t) GetPixelBlue(image,q);
                  SetPixelBlue(image,image->colormap[

ConstrainColormapIndex(image,index,exception)].blue,q);
                }
              SetPixelAlpha(image,image->alpha_trait !=
UndefinedPixelTrait ?
                ScaleCharToQuantum(*(p+number_pixels*3)) :
OpaqueAlpha,q);
              p++;
              q+=GetPixelChannels(image);
            }
            if (SyncAuthenticPixels(image,exception) == MagickFalse)
              break;
            if (image->previous == (Image *) NULL)
              {

status=SetImageProgress(image,LoadImageTag,(MagickOffsetType) y,
                  image->rows);
                if (status == MagickFalse)
                  break;
              }
          }
        }
    pixels=(unsigned char *) RelinquishMagickMemory(pixels);
    if (image->storage_class == PseudoClass)
      (void) SyncImage(image,exception);
    if (EOFBlob(image) != MagickFalse)
      {

ThrowFileException(exception,CorruptImageError,"UnexpectedEndOfFile",
          image->filename);
        break;
      }
    /*
      Proceed to next image.
    */
```

```
    if (image_info->number_scenes != 0)
      if (image->scene >= (image_info->scene+image_info->number_scenes-
1))
        break;
    count=ReadBlob(image,1,&viff_info.identifier);
    if ((count != 0) && (viff_info.identifier == 0xab))
      {
        /*
          Allocate next image structure.
        */
        AcquireNextImage(image_info,image,exception);
        if (GetNextImageInList(image) == (Image *) NULL)
          {
            image=DestroyImageList(image);
            return((Image *) NULL);
          }
        image=SyncNextImageInList(image);
        status=SetImageProgress(image,LoadImagesTag,TellBlob(image),
          GetBlobSize(image));
        if (status == MagickFalse)
          break;
      }
  } while ((count != 0) && (viff_info.identifier == 0xab));
  (void) CloseBlob(image);
  return(GetFirstImageInList(image));
}
<sep>
static int __ip6_append_data(struct sock *sk,
                        struct flowi6 *fl6,
                        struct sk_buff_head *queue,
                        struct inet_cork *cork,
                        struct inet6_cork *v6_cork,
                        struct page_frag *pfrag,
                        int getfrag(void *from, char *to, int offset,
                                int len, int odd, struct sk_buff *skb),
                        void *from, int length, int transhdrlen,
                        unsigned int flags, struct ipcm6_cookie *ipc6,
                        const struct sockcm_cookie *sockc)
{
      struct sk_buff *skb, *skb_prev = NULL;
      unsigned int maxfraglen, fragheaderlen, mtu, orig_mtu;
      int exthdrlen = 0;
      int dst_exthdrlen = 0;
      int hh_len;
      int copy;
      int err;
      int offset = 0;
      __u8 tx_flags = 0;
      u32 tskey = 0;
      struct rt6_info *rt = (struct rt6_info *)cork->dst;
      struct ipv6_txoptions *opt = v6_cork->opt;
      int csummode = CHECKSUM_NONE;
      unsigned int maxnonfragsize, headersize;
```

```
	skb = skb_peek_tail(queue);
	if (!skb) {
		exthdrlen = opt ? opt->opt_flen : 0;
		dst_exthdrlen = rt->dst.header_len - rt->rt6i_nfheader_len;
	}

	mtu = cork->fragsize;
	orig_mtu = mtu;

	hh_len = LL_RESERVED_SPACE(rt->dst.dev);

	fragheaderlen = sizeof(struct ipv6hdr) + rt->rt6i_nfheader_len +
			(opt ? opt->opt_nflen : 0);
	maxfraglen = ((mtu - fragheaderlen) & ~7) + fragheaderlen -
			sizeof(struct frag_hdr);

	headersize = sizeof(struct ipv6hdr) +
			(opt ? opt->opt_flen + opt->opt_nflen : 0) +
			(dst_allfrag(&rt->dst) ?
			 sizeof(struct frag_hdr) : 0) +
			rt->rt6i_nfheader_len;

	if (cork->length + length > mtu - headersize && ipc6->dontfrag &&
	    (sk->sk_protocol == IPPROTO_UDP ||
	     sk->sk_protocol == IPPROTO_RAW)) {
		ipv6_local_rxpmtu(sk, fl6, mtu - headersize +
				  sizeof(struct ipv6hdr));
		goto emsgsize;
	}

	if (ip6_sk_ignore_df(sk))
		maxnonfragsize = sizeof(struct ipv6hdr) + IPV6_MAXPLEN;
	else
		maxnonfragsize = mtu;

	if (cork->length + length > maxnonfragsize - headersize) {
emsgsize:
		ipv6_local_error(sk, EMSGSIZE, fl6,
				 mtu - headersize +
				 sizeof(struct ipv6hdr));
		return -EMSGSIZE;
	}

	/* CHECKSUM_PARTIAL only with no extension headers and when
	 * we are not going to fragment
	 */
	if (transhdrlen && sk->sk_protocol == IPPROTO_UDP &&
	    headersize == sizeof(struct ipv6hdr) &&
	    length <= mtu - headersize &&
	    !(flags & MSG_MORE) &&
	    rt->dst.dev->features & (NETIF_F_IPV6_CSUM | NETIF_F_HW_CSUM))
		csummode = CHECKSUM_PARTIAL;

	if (sk->sk_type == SOCK_DGRAM || sk->sk_type == SOCK_RAW) {
```

```
            sock_tx_timestamp(sk, sockc->tsflags, &tx_flags);
            if (tx_flags & SKBTX_ANY_SW_TSTAMP &&
                sk->sk_tsflags & SOF_TIMESTAMPING_OPT_ID)
                  tskey = sk->sk_tskey++;
      }

      /*
       * Let's try using as much space as possible.
       * Use MTU if total length of the message fits into the MTU.
       * Otherwise, we need to reserve fragment header and
       * fragment alignment (= 8-15 octects, in total).
       *
       * Note that we may need to "move" the data from the tail of
       * of the buffer to the new fragment when we split
       * the message.
       *
       * FIXME: It may be fragmented into multiple chunks
       *        at once if non-fragmentable extension headers
       *        are too large.
       * --yoshfuji
       */

      cork->length += length;
      if ((((length + (skb ? skb->len : headersize)) > mtu) ||
           (skb && skb_is_gso(skb))) &&
          (sk->sk_protocol == IPPROTO_UDP) &&
          (rt->dst.dev->features & NETIF_F_UFO) && !dst_xfrm(&rt->dst) &&
          (sk->sk_type == SOCK_DGRAM) && !udp_get_no_check6_tx(sk)) {
            err = ip6_ufo_append_data(sk, queue, getfrag, from, length,
                                      hh_len, fragheaderlen, exthdrlen,
                                      transhdrlen, mtu, flags, fl6);
            if (err)
                  goto error;
            return 0;
      }

      if (!skb)
            goto alloc_new_skb;

      while (length > 0) {
            /* Check if the remaining data fits into current packet. */
            copy = (cork->length <= mtu && !(cork->flags &
IPCORK_ALLFRAG) ? mtu : maxfraglen) - skb->len;
            if (copy < length)
                  copy = maxfraglen - skb->len;

            if (copy <= 0) {
                  char *data;
                  unsigned int datalen;
                  unsigned int fraglen;
                  unsigned int fraggap;
                  unsigned int alloclen;
alloc_new_skb:
                  /* There's no room in the current skb */
```

```
                if (skb)
                      fraggap = skb->len - maxfraglen;
                else
                      fraggap = 0;
                /* update mtu and maxfraglen if necessary */
                if (!skb || !skb_prev)
                      ip6_append_data_mtu(&mtu, &maxfraglen,
                                          fragheaderlen, skb, rt,
                                          orig_mtu);

                skb_prev = skb;

                /*
                 * If remaining data exceeds the mtu,
                 * we know we need more fragment(s).
                 */
                datalen = length + fraggap;

                if (datalen > (cork->length <= mtu && !(cork->flags &
IPCORK_ALLFRAG) ? mtu : maxfraglen) - fragheaderlen)
                      datalen = maxfraglen - fragheaderlen - rt-
>dst.trailer_len;
                if ((flags & MSG_MORE) &&
                    !(rt->dst.dev->features&NETIF_F_SG))
                      alloclen = mtu;
                else
                      alloclen = datalen + fragheaderlen;

                alloclen += dst_exthdrlen;

                if (datalen != length + fraggap) {
                      /*
                       * this is not the last fragment, the trailer
                       * space is regarded as data space.
                       */
                      datalen += rt->dst.trailer_len;
                }

                alloclen += rt->dst.trailer_len;
                fraglen = datalen + fragheaderlen;

                /*
                 * We just reserve space for fragment header.
                 * Note: this may be overallocation if the message
                 * (without MSG_MORE) fits into the MTU.
                 */
                alloclen += sizeof(struct frag_hdr);

                copy = datalen - transhdrlen - fraggap;
                if (copy < 0) {
                      err = -EINVAL;
                      goto error;
                }
                if (transhdrlen) {
```

```
      skb = sock_alloc_send_skb(sk,
                  alloclen + hh_len,
                  (flags & MSG_DONTWAIT), &err);
} else {
      skb = NULL;
      if (refcount_read(&sk->sk_wmem_alloc) <=
          2 * sk->sk_sndbuf)
            skb = sock_wmalloc(sk,
                           alloclen + hh_len, 1,
                           sk->sk_allocation);
      if (unlikely(!skb))
            err = -ENOBUFS;
}
if (!skb)
      goto error;
/*
 *    Fill in the control structures
 */
skb->protocol = htons(ETH_P_IPV6);
skb->ip_summed = csummode;
skb->csum = 0;
/* reserve for fragmentation and ipsec header */
skb_reserve(skb, hh_len + sizeof(struct frag_hdr) +
          dst_exthdrlen);

/* Only the initial fragment is time stamped */
skb_shinfo(skb)->tx_flags = tx_flags;
tx_flags = 0;
skb_shinfo(skb)->tskey = tskey;
tskey = 0;

/*
 *    Find where to start putting bytes
 */
data = skb_put(skb, fraglen);
skb_set_network_header(skb, exthdrlen);
data += fragheaderlen;
skb->transport_header = (skb->network_header +
                   fragheaderlen);
if (fraggap) {
      skb->csum = skb_copy_and_csum_bits(
            skb_prev, maxfraglen,
            data + transhdrlen, fraggap, 0);
      skb_prev->csum = csum_sub(skb_prev->csum,
                          skb->csum);
      data += fraggap;
      pskb_trim_unique(skb_prev, maxfraglen);
}
if (copy > 0 &&
    getfrag(from, data + transhdrlen, offset,
          copy, fraggap, skb) < 0) {
      err = -EFAULT;
      kfree_skb(skb);
      goto error;
```

```
        }

        offset += copy;
        length -= datalen - fraggap;
        transhdrlen = 0;
        exthdrlen = 0;
        dst_exthdrlen = 0;

        if ((flags & MSG_CONFIRM) && !skb_prev)
                skb_set_dst_pending_confirm(skb, 1);

        /*
         * Put the packet on the pending queue
         */
        __skb_queue_tail(queue, skb);
        continue;
}

if (copy > length)
        copy = length;

if (!(rt->dst.dev->features&NETIF_F_SG)) {
        unsigned int off;

        off = skb->len;
        if (getfrag(from, skb_put(skb, copy),
                        offset, copy, off, skb) < 0) {
                __skb_trim(skb, off);
                err = -EFAULT;
                goto error;
        }
} else {
        int i = skb_shinfo(skb)->nr_frags;

        err = -ENOMEM;
        if (!sk_page_frag_refill(sk, pfrag))
                goto error;

        if (!skb_can_coalesce(skb, i, pfrag->page,
                              pfrag->offset)) {
                err = -EMSGSIZE;
                if (i == MAX_SKB_FRAGS)
                        goto error;

                __skb_fill_page_desc(skb, i, pfrag->page,
                                     pfrag->offset, 0);
                skb_shinfo(skb)->nr_frags = ++i;
                get_page(pfrag->page);
        }
        copy = min_t(int, copy, pfrag->size - pfrag->offset);
        if (getfrag(from,
                    page_address(pfrag->page) + pfrag->offset,
                    offset, copy, skb->len, skb) < 0)
                goto error_efault;
```

```
                pfrag->offset += copy;
                skb_frag_size_add(&skb_shinfo(skb)->frags[i - 1], copy);
                skb->len += copy;
                skb->data_len += copy;
                skb->truesize += copy;
                refcount_add(copy, &sk->sk_wmem_alloc);
            }
            offset += copy;
            length -= copy;
      }

      return 0;

error_efault:
      err = -EFAULT;
error:
      cork->length -= length;
      IP6_INC_STATS(sock_net(sk), rt->rt6i_idev,
IPSTATS_MIB_OUTDISCARDS);
      return err;
}
<sep>
TPMI_RH_ACT_Unmarshal( TPMI_RH_ACT *target, BYTE **buffer, INT32 *size)
{
    TPM_RC rc = TPM_RC_SUCCESS;

    if (rc == TPM_RC_SUCCESS) {
      rc = TPM_HANDLE_Unmarshal(target, buffer, size);
    }
    if (rc == TPM_RC_SUCCESS) {
      BOOL isNotACT = (*target < TPM_RH_ACT_0) || (*target >
TPM_RH_ACT_F);
      if (isNotACT) {
          rc = TPM_RC_VALUE;
      }
    }
    return rc;
}
<sep>
PJ_DEF(int) pj_scan_get_char( pj_scanner *scanner )
{
    int chr = *scanner->curptr;

    if (!chr) {
      pj_scan_syntax_err(scanner);
      return 0;
    }

    ++scanner->curptr;

    if (PJ_SCAN_IS_PROBABLY_SPACE(*scanner->curptr) && scanner->skip_ws)
{
      pj_scan_skip_whitespace(scanner);
```

```
    }
    return chr;
}
<sep>
static int binder_thread_release(struct binder_proc *proc,
                         struct binder_thread *thread)
{
      struct binder_transaction *t;
      struct binder_transaction *send_reply = NULL;
      int active_transactions = 0;
      struct binder_transaction *last_t = NULL;

      binder_inner_proc_lock(thread->proc);
      /*
       * take a ref on the proc so it survives
       * after we remove this thread from proc->threads.
       * The corresponding dec is when we actually
       * free the thread in binder_free_thread()
       */
      proc->tmp_ref++;
      /*
       * take a ref on this thread to ensure it
       * survives while we are releasing it
       */
      atomic_inc(&thread->tmp_ref);
      rb_erase(&thread->rb_node, &proc->threads);
      t = thread->transaction_stack;
      if (t) {
            spin_lock(&t->lock);
            if (t->to_thread == thread)
                  send_reply = t;
      }
      thread->is_dead = true;

      while (t) {
            last_t = t;
            active_transactions++;
            binder_debug(BINDER_DEBUG_DEAD_TRANSACTION,
                         "release %d:%d transaction %d %s, still active\n",
                          proc->pid, thread->pid,
                         t->debug_id,
                         (t->to_thread == thread) ? "in" : "out");

            if (t->to_thread == thread) {
                  t->to_proc = NULL;
                  t->to_thread = NULL;
                  if (t->buffer) {
                        t->buffer->transaction = NULL;
                        t->buffer = NULL;
                  }
                  t = t->to_parent;
            } else if (t->from == thread) {
                  t->from = NULL;
                  t = t->from_parent;
```

```
            } else
                  BUG();
            spin_unlock(&last_t->lock);
            if (t)
                  spin_lock(&t->lock);
      }

      /*
       * If this thread used poll, make sure we remove the waitqueue
       * from any epoll data structures holding it with POLLFREE.
       * waitqueue_active() is safe to use here because we're holding
       * the inner lock.
       */
      if ((thread->looper & BINDER_LOOPER_STATE_POLL) &&
          waitqueue_active(&thread->wait)) {
            wake_up_poll(&thread->wait, EPOLLHUP | POLLFREE);
      }

      binder_inner_proc_unlock(thread->proc);

      if (send_reply)
            binder_send_failed_reply(send_reply, BR_DEAD_REPLY);
      binder_release_work(proc, &thread->todo);
      binder_thread_dec_tmpref(thread);
      return active_transactions;
}
<sep>
static void ahash_op_unaligned_finish(struct ahash_request *req, int err)
{
      struct ahash_request_priv *priv = req->priv;

      if (err == -EINPROGRESS)
            return;

      if (!err)
            memcpy(priv->result, req->result,
                   crypto_ahash_digestsize(crypto_ahash_reqtfm(req)));

      ahash_restore_req(req);
}
<sep>
static void p54u_disconnect(struct usb_interface *intf)
{
      struct ieee80211_hw *dev = usb_get_intfdata(intf);
      struct p54u_priv *priv;

      if (!dev)
            return;

      priv = dev->priv;
      wait_for_completion(&priv->fw_wait_load);
      p54_unregister_common(dev);

      usb_put_dev(interface_to_usbdev(intf));
```

```
      release_firmware(priv->fw);
      p54_free_common(dev);
}
<sep>
int mongo_env_read_socket( mongo *conn, void *buf, int len ) {
    char *cbuf = buf;
    while ( len ) {
        int sent = recv( conn->sock, cbuf, len, 0 );
        if ( sent == 0 || sent == -1 ) {
            __mongo_set_error( conn, MONGO_IO_ERROR, strerror( errno ),
errno );
            return MONGO_ERROR;
        }
        cbuf += sent;
        len -= sent;
    }

    return MONGO_OK;
}
<sep>
static int asn1_template_noexp_d2i(ASN1_VALUE **val,
                                   const unsigned char **in, long len,
                                   const ASN1_TEMPLATE *tt, char opt,
                                   ASN1_TLC *ctx)
{
    int flags, aclass;
    int ret;
    const unsigned char *p, *q;
    if (!val)
        return 0;
    flags = tt->flags;
    aclass = flags & ASN1_TFLG_TAG_CLASS;

    p = *in;
    q = p;

    if (flags & ASN1_TFLG_SK_MASK) {
        /* SET OF, SEQUENCE OF */
        int sktag, skaclass;
        char sk_eoc;
        /* First work out expected inner tag value */
        if (flags & ASN1_TFLG_IMPTAG) {
            sktag = tt->tag;
            skaclass = aclass;
        } else {
            skaclass = V_ASN1_UNIVERSAL;
            if (flags & ASN1_TFLG_SET_OF)
                sktag = V_ASN1_SET;
            else
                sktag = V_ASN1_SEQUENCE;
        }
        /* Get the tag */
        ret = asn1_check_tlen(&len, NULL, NULL, &sk_eoc, NULL,
                              &p, len, sktag, skaclass, opt, ctx);
```

```
        if (!ret) {
            ASN1err(ASN1_F_ASN1_TEMPLATE_NOEXP_D2I,
ERR_R_NESTED_ASN1_ERROR);
            return 0;
        } else if (ret == -1)
            return -1;
        if (!*val)
            *val = (ASN1_VALUE *)sk_new_null();
        else {
            /*
             * We've got a valid STACK: free up any items present
             */
            STACK_OF(ASN1_VALUE) *sktmp = (STACK_OF(ASN1_VALUE) *)*val;
            ASN1_VALUE *vtmp;
            while (sk_ASN1_VALUE_num(sktmp) > 0) {
                vtmp = sk_ASN1_VALUE_pop(sktmp);
                ASN1_item_ex_free(&vtmp, ASN1_ITEM_ptr(tt->item));
            }
        }

        if (!*val) {
            ASN1err(ASN1_F_ASN1_TEMPLATE_NOEXP_D2I,
ERR_R_MALLOC_FAILURE);
            goto err;
        }

        /* Read as many items as we can */
        while (len > 0) {
            ASN1_VALUE *skfield;
            q = p;
            /* See if EOC found */
            if (asn1_check_eoc(&p, len)) {
                if (!sk_eoc) {
                    ASN1err(ASN1_F_ASN1_TEMPLATE_NOEXP_D2I,
                            ASN1_R_UNEXPECTED_EOC);
                    goto err;
                }
                len -= p - q;
                sk_eoc = 0;
                break;
            }
            skfield = NULL;
            if (!ASN1_item_ex_d2i(&skfield, &p, len,
                                  ASN1_ITEM_ptr(tt->item), -1, 0, 0,
ctx)) {
                ASN1err(ASN1_F_ASN1_TEMPLATE_NOEXP_D2I,
                        ERR_R_NESTED_ASN1_ERROR);
                goto err;
            }
            len -= p - q;
            if (!sk_ASN1_VALUE_push((STACK_OF(ASN1_VALUE) *)*val,
skfield)) {
                ASN1err(ASN1_F_ASN1_TEMPLATE_NOEXP_D2I,
ERR_R_MALLOC_FAILURE);
```

```
                goto err;
            }
        }
        if (sk_eoc) {
            ASN1err(ASN1_F_ASN1_TEMPLATE_NOEXP_D2I, ASN1_R_MISSING_EOC);
            goto err;
        }
    } else if (flags & ASN1_TFLG_IMPTAG) {
        /* IMPLICIT tagging */
        ret = ASN1_item_ex_d2i(val, &p, len,
                               ASN1_ITEM_ptr(tt->item), tt->tag, aclass,
opt,
                               ctx);
        if (!ret) {
            ASN1err(ASN1_F_ASN1_TEMPLATE_NOEXP_D2I,
ERR_R_NESTED_ASN1_ERROR);
            goto err;
        } else if (ret == -1)
            return -1;
    } else {
        /* Nothing special */
        ret = ASN1_item_ex_d2i(val, &p, len, ASN1_ITEM_ptr(tt->item),
                               -1, 0, opt, ctx);
        if (!ret) {
            ASN1err(ASN1_F_ASN1_TEMPLATE_NOEXP_D2I,
ERR_R_NESTED_ASN1_ERROR);
            goto err;
        } else if (ret == -1)
            return -1;
    }

    *in = p;
    return 1;

 err:
    ASN1_template_free(val, tt);
    return 0;
}
<sep>
PHP_HASH_API void PHP_HAVAL256Final(unsigned char *digest, PHP_HAVAL_CTX
* context)
{
      unsigned char bits[10];
      unsigned int index, padLen;

      /* Version, Passes, and Digest Length */
      bits[0] =   (PHP_HASH_HAVAL_VERSION & 0x07) |
                       ((context->passes & 0x07) << 3) |
                       ((context->output & 0x03) << 6);
      bits[1] = (context->output >> 2);

      /* Save number of bits */
      Encode(bits + 2, context->count, 8);
```

```
      /* Pad out to 118 mod 128.
       */
      index = (unsigned int) ((context->count[0] >> 3) & 0x3f);
      padLen = (index < 118) ? (118 - index) : (246 - index);
      PHP_HAVALUpdate(context, PADDING, padLen);

      /* Append version, passes, digest length, and message length */
      PHP_HAVALUpdate(context, bits, 10);

      /* Store state in digest */
      Encode(digest, context->state, 32);

      /* Zeroize sensitive information.
       */
      memset((unsigned char*) context, 0, sizeof(*context));
}
<sep>
IPV6BuildTestPacket(uint32_t id, uint16_t off, int mf, const char
content,
    int content_len)
{
    Packet *p = NULL;
    uint8_t *pcontent;
    IPV6Hdr ip6h;

    p = SCCalloc(1, sizeof(*p) + default_packet_size);
    if (unlikely(p == NULL))
        return NULL;

    PACKET_INITIALIZE(p);

    gettimeofday(&p->ts, NULL);

    ip6h.s_ip6_nxt = 44;
    ip6h.s_ip6_hlim = 2;

    /* Source and dest address - very bogus addresses. */
    ip6h.s_ip6_src[0] = 0x01010101;
    ip6h.s_ip6_src[1] = 0x01010101;
    ip6h.s_ip6_src[2] = 0x01010101;
    ip6h.s_ip6_src[3] = 0x01010101;
    ip6h.s_ip6_dst[0] = 0x02020202;
    ip6h.s_ip6_dst[1] = 0x02020202;
    ip6h.s_ip6_dst[2] = 0x02020202;
    ip6h.s_ip6_dst[3] = 0x02020202;

    /* copy content_len crap, we need full length */
    PacketCopyData(p, (uint8_t *)&ip6h, sizeof(IPV6Hdr));

    p->ip6h = (IPV6Hdr *)GET_PKT_DATA(p);
    IPV6_SET_RAW_VER(p->ip6h, 6);
    /* Fragmentation header. */
    IPV6FragHdr *fh = (IPV6FragHdr *)(GET_PKT_DATA(p) + sizeof(IPV6Hdr));
    fh->ip6fh_nxt = IPPROTO_ICMP;
```

```
    fh->ip6fh_ident = htonl(id);
    fh->ip6fh_offlg = htons((off << 3) | mf);

    DecodeIPV6FragHeader(p, (uint8_t *)fh, 8, 8 + content_len, 0);

    pcontent = SCCalloc(1, content_len);
    if (unlikely(pcontent == NULL))
        return NULL;
    memset(pcontent, content, content_len);
    PacketCopyDataOffset(p, sizeof(IPV6Hdr) + sizeof(IPV6FragHdr),
pcontent, content_len);
    SET_PKT_LEN(p, sizeof(IPV6Hdr) + sizeof(IPV6FragHdr) + content_len);
    SCFree(pcontent);

    p->ip6h->s_ip6_plen = htons(sizeof(IPV6FragHdr) + content_len);

    SET_IPV6_SRC_ADDR(p, &p->src);
    SET_IPV6_DST_ADDR(p, &p->dst);

    /* Self test. */
    if (IPV6_GET_VER(p) != 6)
        goto error;
    if (IPV6_GET_NH(p) != 44)
        goto error;
    if (IPV6_GET_PLEN(p) != sizeof(IPV6FragHdr) + content_len)
        goto error;

    return p;
error:
    fprintf(stderr, "Error building test packet.\n");
    if (p != NULL)
        SCFree(p);
    return NULL;
}
<sep>
int main(int argc, char ** argv)
{
      int c;
      unsigned long flags = MS_MANDLOCK;
      char * orgoptions = NULL;
      char * share_name = NULL;
      const char * ipaddr = NULL;
      char * uuid = NULL;
      char * mountpoint = NULL;
      char * options = NULL;
      char * optionstail;
      char * resolved_path = NULL;
      char * temp;
      char * dev_name;
      int rc = 0;
      int rsize = 0;
      int wsize = 0;
      int nomtab = 0;
      int uid = 0;
```

```
	int gid = 0;
	int optlen = 0;
	int orgoptlen = 0;
	size_t options_size = 0;
	size_t current_len;
	int retry = 0; /* set when we have to retry mount with uppercase */
	struct addrinfo *addrhead = NULL, *addr;
	struct utsname sysinfo;
	struct mntent mountent;
	struct sockaddr_in *addr4;
	struct sockaddr_in6 *addr6;
	FILE * pmntfile;

	/* setlocale(LC_ALL, "");
	bindtextdomain(PACKAGE, LOCALEDIR);
	textdomain(PACKAGE); */

	if(argc && argv)
		thisprogram = argv[0];
	else
		mount_cifs_usage(stderr);

	if(thisprogram == NULL)
		thisprogram = "mount.cifs";

	uname(&sysinfo);
	/* BB add workstation name and domain and pass down */

/* #ifdef _GNU_SOURCE
	fprintf(stderr, " node: %s machine: %s sysname %s domain %s\n",
sysinfo.nodename,sysinfo.machine,sysinfo.sysname,sysinfo.domainname);
#endif */
	if(argc > 2) {
		dev_name = argv[1];
		share_name = strndup(argv[1], MAX_UNC_LEN);
		if (share_name == NULL) {
			fprintf(stderr, "%s: %s", argv[0], strerror(ENOMEM));
			exit(EX_SYSERR);
		}
		mountpoint = argv[2];
	} else if (argc == 2) {
		if ((strcmp(argv[1], "-V") == 0) ||
		    (strcmp(argv[1], "--version") == 0))
		{
			print_cifs_mount_version();
			exit(0);
		}

		if ((strcmp(argv[1], "-h") == 0) ||
		    (strcmp(argv[1], "-?") == 0) ||
		    (strcmp(argv[1], "--help") == 0))
			mount_cifs_usage(stdout);

		mount_cifs_usage(stderr);
```

```
	} else {
		mount_cifs_usage(stderr);
	}


	/* add sharename in opts string as unc= parm */
	while ((c = getopt_long (argc, argv, "afFhilL:no:O:rsSU:vVwt:",
			longopts, NULL)) != -1) {
		switch (c) {
/* No code to do the following  options yet */
/*	case 'l':
		list_with_volumelabel = 1;
		break;
	case 'L':
		volumelabel = optarg;
		break; */
/*	case 'a':
		++mount_all;
		break; */

		case '?':
		case 'h':	/* help */
			mount_cifs_usage(stdout);
		case 'n':
			++nomtab;
			break;
		case 'b':
#ifdef MS_BIND
			flags |= MS_BIND;
#else
			fprintf(stderr,
				"option 'b' (MS_BIND) not supported\n");
#endif
			break;
		case 'm':
#ifdef MS_MOVE
			flags |= MS_MOVE;
#else
			fprintf(stderr,
				"option 'm' (MS_MOVE) not supported\n");
#endif
			break;
		case 'o':
			orgoptions = strdup(optarg);
		    break;
		case 'r':  /* mount readonly */
			flags |= MS_RDONLY;
			break;
		case 'U':
			uuid = optarg;
			break;
		case 'v':
			++verboseflag;
			break;
```

```
            case 'V':
                  print_cifs_mount_version();
                  exit (0);
            case 'w':
                  flags &= ~MS_RDONLY;
                  break;
            case 'R':
                  rsize = atoi(optarg) ;
                  break;
            case 'W':
                  wsize = atoi(optarg);
                  break;
            case '1':
                  if (isdigit(*optarg)) {
                        char *ep;

                        uid = strtoul(optarg, &ep, 10);
                        if (*ep) {
                              fprintf(stderr, "bad uid value \"%s\"\n",
optarg);
                              exit(EX_USAGE);
                        }
                  } else {
                        struct passwd *pw;

                        if (!(pw = getpwnam(optarg))) {
                              fprintf(stderr, "bad user name \"%s\"\n",
optarg);
                              exit(EX_USAGE);
                        }
                        uid = pw->pw_uid;
                        endpwent();
                  }
                  break;
            case '2':
                  if (isdigit(*optarg)) {
                        char *ep;

                        gid = strtoul(optarg, &ep, 10);
                        if (*ep) {
                              fprintf(stderr, "bad gid value \"%s\"\n",
optarg);
                              exit(EX_USAGE);
                        }
                  } else {
                        struct group *gr;

                        if (!(gr = getgrnam(optarg))) {
                              fprintf(stderr, "bad user name \"%s\"\n",
optarg);
                              exit(EX_USAGE);
                        }
                        gid = gr->gr_gid;
                        endpwent();
```

```
                }
                break;
            case 'u':
                got_user = 1;
                user_name = optarg;
                break;
            case 'd':
                domain_name = optarg; /* BB fix this - currently ignored
*/
                got_domain = 1;
                break;
            case 'p':
                if(mountpassword == NULL)
                    mountpassword = (char
*)calloc(MOUNT_PASSWD_SIZE+1,1);
                if(mountpassword) {
                    got_password = 1;
                    strlcpy(mountpassword,optarg,MOUNT_PASSWD_SIZE+1);
                }
                break;
            case 'S':
                get_password_from_file(0 /* stdin */,NULL);
                break;
            case 't':
                break;
            case 'f':
                ++fakemnt;
                break;
            default:
                fprintf(stderr, "unknown mount option %c\n",c);
                mount_cifs_usage(stderr);
            }
        }

        if((argc < 3) || (dev_name == NULL) || (mountpoint == NULL)) {
            mount_cifs_usage(stderr);
        }

        /* make sure mountpoint is legit */
        rc = chdir(mountpoint);
        if (rc) {
            fprintf(stderr, "Couldn't chdir to %s: %s\n", mountpoint,
                    strerror(errno));
            rc = EX_USAGE;
            goto mount_exit;
        }

        rc = check_mountpoint(thisprogram, mountpoint);
        if (rc)
            goto mount_exit;

        /* sanity check for unprivileged mounts */
        if (getuid()) {
            rc = check_fstab(thisprogram, mountpoint, dev_name,
```

```
                        &orgoptions);
            if (rc)
                  goto mount_exit;

            /* enable any default user mount flags */
            flags |= CIFS_SETUID_FLAGS;
      }

      if (getenv("PASSWD")) {
            if(mountpassword == NULL)
                  mountpassword = (char *)calloc(MOUNT_PASSWD_SIZE+1,1);
            if(mountpassword) {

      strlcpy(mountpassword,getenv("PASSWD"),MOUNT_PASSWD_SIZE+1);
                  got_password = 1;
            }
      } else if (getenv("PASSWD_FD")) {
            get_password_from_file(atoi(getenv("PASSWD_FD")),NULL);
      } else if (getenv("PASSWD_FILE")) {
            get_password_from_file(0, getenv("PASSWD_FILE"));
      }

        if (orgoptions && parse_options(&orgoptions, &flags)) {
                rc = EX_USAGE;
            goto mount_exit;
      }

      if (getuid()) {
#if !CIFS_LEGACY_SETUID_CHECK
            if (!(flags & (MS_USERS|MS_USER))) {
                  fprintf(stderr, "%s: permission denied\n", thisprogram);
                  rc = EX_USAGE;
                  goto mount_exit;
            }
#endif /* !CIFS_LEGACY_SETUID_CHECK */

            if (geteuid()) {
                  fprintf(stderr, "%s: not installed setuid - \"user\" "
                              "CIFS mounts not supported.",
                              thisprogram);
                  rc = EX_FAIL;
                  goto mount_exit;
            }
      }

      flags &= ~(MS_USERS|MS_USER);

      addrhead = addr = parse_server(&share_name);
      if((addrhead == NULL) && (got_ip == 0)) {
            fprintf(stderr, "No ip address specified and hostname not
found\n");
            rc = EX_USAGE;
            goto mount_exit;
      }
```

```
      /* BB save off path and pop after mount returns? */
      resolved_path = (char *)malloc(PATH_MAX+1);
      if (!resolved_path) {
            fprintf(stderr, "Unable to allocate memory.\n");
            rc = EX_SYSERR;
            goto mount_exit;
      }

      /* Note that if we can not canonicalize the name, we get
         another chance to see if it is valid when we chdir to it */
      if(!realpath(".", resolved_path)) {
            fprintf(stderr, "Unable to resolve %s to canonical path:
%s\n",
                        mountpoint, strerror(errno));
            rc = EX_SYSERR;
            goto mount_exit;
      }

      mountpoint = resolved_path;

      if(got_user == 0) {
            /* Note that the password will not be retrieved from the
               USER env variable (ie user%password form) as there is
               already a PASSWD environment varaible */
            if (getenv("USER"))
                  user_name = strdup(getenv("USER"));
            if (user_name == NULL)
                  user_name = getusername();
            got_user = 1;
      }

      if(got_password == 0) {
            char *tmp_pass = getpass("Password: "); /* BB obsolete sys
call but
                                             no good replacement yet. */
            mountpassword = (char *)calloc(MOUNT_PASSWD_SIZE+1,1);
            if (!tmp_pass || !mountpassword) {
                  fprintf(stderr, "Password not entered, exiting\n");
                  exit(EX_USAGE);
            }
            strlcpy(mountpassword, tmp_pass, MOUNT_PASSWD_SIZE+1);
            got_password = 1;
      }
      /* FIXME launch daemon (handles dfs name resolution and credential
change)
         remember to clear parms and overwrite password field before
launching */
      if(orgoptions) {
            optlen = strlen(orgoptions);
            orgoptlen = optlen;
      } else
            optlen = 0;
      if(share_name)
```

```
            optlen += strlen(share_name) + 4;
      else {
            fprintf(stderr, "No server share name specified\n");
            fprintf(stderr, "\nMounting the DFS root for server not
implemented yet\n");
                exit(EX_USAGE);
      }
      if(user_name)
            optlen += strlen(user_name) + 6;
      optlen += MAX_ADDRESS_LEN + 4;
      if(mountpassword)
            optlen += strlen(mountpassword) + 6;
mount_retry:
      SAFE_FREE(options);
      options_size = optlen + 10 + DOMAIN_SIZE;
      options = (char *)malloc(options_size /* space for commas in
password */ + 8 /* space for domain=  , domain name itself was counted as
part of the length username string above */);

      if(options == NULL) {
            fprintf(stderr, "Could not allocate memory for mount
options\n");
            exit(EX_SYSERR);
      }

      strlcpy(options, "unc=", options_size);
      strlcat(options,share_name,options_size);
      /* scan backwards and reverse direction of slash */
      temp = strrchr(options, '/');
      if(temp > options + 6)
            *temp = '\\';
      if(user_name) {
            /* check for syntax like user=domain\user */
            if(got_domain == 0)
                  domain_name = check_for_domain(&user_name);
            strlcat(options,",user=",options_size);
            strlcat(options,user_name,options_size);
      }
      if(retry == 0) {
            if(domain_name) {
                  /* extra length accounted for in option string above */
                  strlcat(options,",domain=",options_size);
                  strlcat(options,domain_name,options_size);
            }
      }

      strlcat(options,",ver=",options_size);
      strlcat(options,MOUNT_CIFS_VERSION_MAJOR,options_size);

      if(orgoptions) {
            strlcat(options,",",options_size);
            strlcat(options,orgoptions,options_size);
      }
      if(prefixpath) {
```

```
            strlcat(options,",prefixpath=",options_size);
            strlcat(options,prefixpath,options_size); /* no need to cat
the / */
      }

      /* convert all '\\' to '/' in share portion so that /proc/mounts
looks pretty */
      replace_char(dev_name, '\\', '/', strlen(share_name));

      if (!got_ip && addr) {
            strlcat(options, ",ip=", options_size);
            current_len = strnlen(options, options_size);
            optionstail = options + current_len;
            switch (addr->ai_addr->sa_family) {
            case AF_INET6:
                  addr6 = (struct sockaddr_in6 *) addr->ai_addr;
                  ipaddr = inet_ntop(AF_INET6, &addr6->sin6_addr,
optionstail,
                                    options_size - current_len);
                  break;
            case AF_INET:
                  addr4 = (struct sockaddr_in *) addr->ai_addr;
                  ipaddr = inet_ntop(AF_INET, &addr4->sin_addr,
optionstail,
                                    options_size - current_len);
                  break;
            default:
                  ipaddr = NULL;
            }

            /* if the address looks bogus, try the next one */
            if (!ipaddr) {
                  addr = addr->ai_next;
                  if (addr)
                        goto mount_retry;
                  rc = EX_SYSERR;
                  goto mount_exit;
            }
      }

      if (addr->ai_addr->sa_family == AF_INET6 && addr6->sin6_scope_id) {
            strlcat(options, "%", options_size);
            current_len = strnlen(options, options_size);
            optionstail = options + current_len;
            snprintf(optionstail, options_size - current_len, "%u",
                   addr6->sin6_scope_id);
      }

      if(verboseflag)
            fprintf(stderr, "\nmount.cifs kernel mount options: %s",
options);

      if (mountpassword) {
            /*
```

```
             * Commas have to be doubled, or else they will
             * look like the parameter separator
             */
            if(retry == 0)
                  check_for_comma(&mountpassword);
            strlcat(options,",pass=",options_size);
            strlcat(options,mountpassword,options_size);
            if (verboseflag)
                  fprintf(stderr, ",pass=********");
      }

      if (verboseflag)
            fprintf(stderr, "\n");

      rc = check_mtab(thisprogram, dev_name, mountpoint);
      if (rc)
            goto mount_exit;

      if (!fakemnt && mount(dev_name, ".", cifs_fstype, flags, options))
{
            switch (errno) {
            case ECONNREFUSED:
            case EHOSTUNREACH:
                  if (addr) {
                        addr = addr->ai_next;
                        if (addr)
                              goto mount_retry;
                  }
                  break;
            case ENODEV:
                  fprintf(stderr, "mount error: cifs filesystem not
supported by the system\n");
                  break;
            case ENXIO:
                  if(retry == 0) {
                        retry = 1;
                        if (uppercase_string(dev_name) &&
                            uppercase_string(share_name) &&
                            uppercase_string(prefixpath)) {
                              fprintf(stderr, "retrying with upper case
share name\n");
                              goto mount_retry;
                        }
                  }
            }
            fprintf(stderr, "mount error(%d): %s\n", errno,
strerror(errno));
            fprintf(stderr, "Refer to the mount.cifs(8) manual page (e.g.
man "
                   "mount.cifs)\n");
            rc = EX_FAIL;
            goto mount_exit;
      }
```

```
	if (nomtab)
		goto mount_exit;
	atexit(unlock_mtab);
	rc = lock_mtab();
	if (rc) {
		fprintf(stderr, "cannot lock mtab");
		goto mount_exit;
	}
	pmntfile = setmntent(MOUNTED, "a+");
	if (!pmntfile) {
		fprintf(stderr, "could not update mount table\n");
		unlock_mtab();
		rc = EX_FILEIO;
		goto mount_exit;
	}
	mountent.mnt_fsname = dev_name;
	mountent.mnt_dir = mountpoint;
	mountent.mnt_type = (char *)(void *)cifs_fstype;
	mountent.mnt_opts = (char *)malloc(220);
	if(mountent.mnt_opts) {
		char * mount_user = getusername();
		memset(mountent.mnt_opts,0,200);
		if(flags & MS_RDONLY)
			strlcat(mountent.mnt_opts,"ro",220);
		else
			strlcat(mountent.mnt_opts,"rw",220);
		if(flags & MS_MANDLOCK)
			strlcat(mountent.mnt_opts,",mand",220);
		if(flags & MS_NOEXEC)
			strlcat(mountent.mnt_opts,",noexec",220);
		if(flags & MS_NOSUID)
			strlcat(mountent.mnt_opts,",nosuid",220);
		if(flags & MS_NODEV)
			strlcat(mountent.mnt_opts,",nodev",220);
		if(flags & MS_SYNCHRONOUS)
			strlcat(mountent.mnt_opts,",sync",220);
		if(mount_user) {
			if(getuid() != 0) {
				strlcat(mountent.mnt_opts,
					",user=", 220);
				strlcat(mountent.mnt_opts,
					mount_user, 220);
			}
		}
	}
	mountent.mnt_freq = 0;
	mountent.mnt_passno = 0;
	rc = addmntent(pmntfile,&mountent);
	endmntent(pmntfile);
	unlock_mtab();
	SAFE_FREE(mountent.mnt_opts);
	if (rc)
		rc = EX_FILEIO;
mount_exit:
```

```
      if(mountpassword) {
            int len = strlen(mountpassword);
            memset(mountpassword,0,len);
            SAFE_FREE(mountpassword);
      }

      if (addrhead)
            freeaddrinfo(addrhead);
      SAFE_FREE(options);
      SAFE_FREE(orgoptions);
      SAFE_FREE(resolved_path);
      SAFE_FREE(share_name);
      exit(rc);
}
<sep>
MagickExport void DestroyXResources(void)
{
  register int
    i;

  unsigned int
    number_windows;

  XWindowInfo
    *magick_windows[MaxXWindows];

  XWindows
    *windows;

  DestroyXWidget();
  windows=XSetWindows((XWindows *) ~0);
  if ((windows == (XWindows *) NULL) || (windows->display == (Display *)
NULL))
    return;
  number_windows=0;
  magick_windows[number_windows++]=(&windows->context);
  magick_windows[number_windows++]=(&windows->group_leader);
  magick_windows[number_windows++]=(&windows->backdrop);
  magick_windows[number_windows++]=(&windows->icon);
  magick_windows[number_windows++]=(&windows->image);
  magick_windows[number_windows++]=(&windows->info);
  magick_windows[number_windows++]=(&windows->magnify);
  magick_windows[number_windows++]=(&windows->pan);
  magick_windows[number_windows++]=(&windows->command);
  magick_windows[number_windows++]=(&windows->widget);
  magick_windows[number_windows++]=(&windows->popup);
  for (i=0; i < (int) number_windows; i++)
  {
    if (magick_windows[i]->mapped != MagickFalse)
      {
        (void) XWithdrawWindow(windows->display,magick_windows[i]->id,
          magick_windows[i]->screen);
        magick_windows[i]->mapped=MagickFalse;
      }
```

```
    if (magick_windows[i]->name != (char *) NULL)
      magick_windows[i]->name=(char *)
        RelinquishMagickMemory(magick_windows[i]->name);
    if (magick_windows[i]->icon_name != (char *) NULL)
      magick_windows[i]->icon_name=(char *)
        RelinquishMagickMemory(magick_windows[i]->icon_name);
    if (magick_windows[i]->cursor != (Cursor) NULL)
      {
        (void) XFreeCursor(windows->display,magick_windows[i]->cursor);
        magick_windows[i]->cursor=(Cursor) NULL;
      }
    if (magick_windows[i]->busy_cursor != (Cursor) NULL)
      {
        (void) XFreeCursor(windows->display,magick_windows[i]-
>busy_cursor);
        magick_windows[i]->busy_cursor=(Cursor) NULL;
      }
    if (magick_windows[i]->highlight_stipple != (Pixmap) NULL)
      {
        (void) XFreePixmap(windows->display,
          magick_windows[i]->highlight_stipple);
        magick_windows[i]->highlight_stipple=(Pixmap) NULL;
      }
    if (magick_windows[i]->shadow_stipple != (Pixmap) NULL)
      {
        (void) XFreePixmap(windows->display,magick_windows[i]-
>shadow_stipple);
        magick_windows[i]->shadow_stipple=(Pixmap) NULL;
      }
    if (magick_windows[i]->ximage != (XImage *) NULL)
      {
        XDestroyImage(magick_windows[i]->ximage);
        magick_windows[i]->ximage=(XImage *) NULL;
      }
    if (magick_windows[i]->pixmap != (Pixmap) NULL)
      {
        (void) XFreePixmap(windows->display,magick_windows[i]->pixmap);
        magick_windows[i]->pixmap=(Pixmap) NULL;
      }
    if (magick_windows[i]->id != (Window) NULL)
      {
        (void) XDestroyWindow(windows->display,magick_windows[i]->id);
        magick_windows[i]->id=(Window) NULL;
      }
    if (magick_windows[i]->destroy != MagickFalse)
      {
        if (magick_windows[i]->image != (Image *) NULL)
          {
            magick_windows[i]->image=DestroyImage(magick_windows[i]-
>image);
            magick_windows[i]->image=NewImageList();
          }
        if (magick_windows[i]->matte_pixmap != (Pixmap) NULL)
          {
```

```
            (void) XFreePixmap(windows->display,
              magick_windows[i]->matte_pixmap);
            magick_windows[i]->matte_pixmap=(Pixmap) NULL;
          }
      }
    if (magick_windows[i]->segment_info != (void *) NULL)
      {
#if defined(MAGICKCORE_HAVE_SHARED_MEMORY)
        XShmSegmentInfo
          *segment_info;

        segment_info=(XShmSegmentInfo *) magick_windows[i]->segment_info;
        if (segment_info != (XShmSegmentInfo *) NULL)
          if (segment_info[0].shmid >= 0)
            {
              if (segment_info[0].shmaddr != NULL)
                (void) shmdt(segment_info[0].shmaddr);
              (void) shmctl(segment_info[0].shmid,IPC_RMID,0);
              segment_info[0].shmaddr=NULL;
              segment_info[0].shmid=(-1);
            }
#endif
        magick_windows[i]->segment_info=(void *) RelinquishMagickMemory(
          magick_windows[i]->segment_info);
      }
  }
  windows->icon_resources=(XResourceInfo *)
    RelinquishMagickMemory(windows->icon_resources);
  if (windows->icon_pixel != (XPixelInfo *) NULL)
    {
      if (windows->icon_pixel->pixels != (unsigned long *) NULL)
        windows->icon_pixel->pixels=(unsigned long *)
          RelinquishMagickMemory(windows->icon_pixel->pixels);
      if (windows->icon_pixel->annotate_context != (GC) NULL)
        XFreeGC(windows->display,windows->icon_pixel->annotate_context);
      windows->icon_pixel=(XPixelInfo *)
        RelinquishMagickMemory(windows->icon_pixel);
    }
  if (windows->pixel_info != (XPixelInfo *) NULL)
    {
      if (windows->pixel_info->pixels != (unsigned long *) NULL)
        windows->pixel_info->pixels=(unsigned long *)
          RelinquishMagickMemory(windows->pixel_info->pixels);
      if (windows->pixel_info->annotate_context != (GC) NULL)
        XFreeGC(windows->display,windows->pixel_info->annotate_context);
      if (windows->pixel_info->widget_context != (GC) NULL)
        XFreeGC(windows->display,windows->pixel_info->widget_context);
      if (windows->pixel_info->highlight_context != (GC) NULL)
        XFreeGC(windows->display,windows->pixel_info->highlight_context);
      windows->pixel_info=(XPixelInfo *)
        RelinquishMagickMemory(windows->pixel_info);
    }
  if (windows->font_info != (XFontStruct *) NULL)
    {
```

```
      XFreeFont(windows->display,windows->font_info);
      windows->font_info=(XFontStruct *) NULL;
    }
  if (windows->class_hints != (XClassHint *) NULL)
    {
      if (windows->class_hints->res_name != (char *) NULL)
        windows->class_hints->res_name=DestroyString(
          windows->class_hints->res_name);
      if (windows->class_hints->res_class != (char *) NULL)
        windows->class_hints->res_class=DestroyString(
          windows->class_hints->res_class);
      XFree(windows->class_hints);
      windows->class_hints=(XClassHint *) NULL;
    }
  if (windows->manager_hints != (XWMHints *) NULL)
    {
      XFree(windows->manager_hints);
      windows->manager_hints=(XWMHints *) NULL;
    }
  if (windows->map_info != (XStandardColormap *) NULL)
    {
      XFree(windows->map_info);
      windows->map_info=(XStandardColormap *) NULL;
    }
  if (windows->icon_map != (XStandardColormap *) NULL)
    {
      XFree(windows->icon_map);
      windows->icon_map=(XStandardColormap *) NULL;
    }
  if (windows->visual_info != (XVisualInfo *) NULL)
    {
      XFree(windows->visual_info);
      windows->visual_info=(XVisualInfo *) NULL;
    }
  if (windows->icon_visual != (XVisualInfo *) NULL)
    {
      XFree(windows->icon_visual);
      windows->icon_visual=(XVisualInfo *) NULL;
    }
  (void) XSetWindows((XWindows *) NULL);
}
<sep>
static int hidp_add_connection(struct input_device *idev)
{
      struct hidp_connadd_req *req;
      sdp_record_t *rec;
      char src_addr[18], dst_addr[18];
      char filename[PATH_MAX];
      GKeyFile *key_file;
      char handle[11], *str;
      GError *gerr = NULL;
      int err;

      req = g_new0(struct hidp_connadd_req, 1);
```

```
	req->ctrl_sock = g_io_channel_unix_get_fd(idev->ctrl_io);
	req->intr_sock = g_io_channel_unix_get_fd(idev->intr_io);
	req->flags     = 0;
	req->idle_to   = idle_timeout;

	ba2str(&idev->src, src_addr);
	ba2str(&idev->dst, dst_addr);

	snprintf(filename, PATH_MAX, STORAGEDIR "/%s/cache/%s", src_addr,
                                                dst_addr);
	sprintf(handle, "0x%8.8X", idev->handle);

	key_file = g_key_file_new();
	g_key_file_load_from_file(key_file, filename, 0, NULL);
	str = g_key_file_get_string(key_file, "ServiceRecords", handle,
NULL);
	g_key_file_free(key_file);

	if (!str) {
		error("Rejected connection from unknown device %s",
dst_addr);
		err = -EPERM;
		goto cleanup;
	}

	rec = record_from_string(str);
	g_free(str);

	err = extract_hid_record(rec, req);
	sdp_record_free(rec);
	if (err < 0) {
		error("Could not parse HID SDP record: %s (%d)", strerror(-
err),
                                                        -err);
		goto cleanup;
	}

	req->vendor = btd_device_get_vendor(idev->device);
	req->product = btd_device_get_product(idev->device);
	req->version = btd_device_get_version(idev->device);

	if (device_name_known(idev->device))
		device_get_name(idev->device, req->name, sizeof(req->name));

	/* Encryption is mandatory for keyboards */
	if (req->subclass & 0x40) {
		if (!bt_io_set(idev->intr_io, &gerr,
                               BT_IO_OPT_SEC_LEVEL, BT_IO_SEC_MEDIUM,
                               BT_IO_OPT_INVALID)) {
			error("btio: %s", gerr->message);
			g_error_free(gerr);
			err = -EFAULT;
			goto cleanup;
		}
```

```
            idev->req = req;
            idev->sec_watch = g_io_add_watch(idev->intr_io, G_IO_OUT,
                                             encrypt_notify, idev);

            return 0;
      }

      if (idev->uhid)
            err = uhid_connadd(idev, req);
      else
            err = ioctl_connadd(req);

cleanup:
      g_free(req->rd_data);
      g_free(req);

      return err;
}
<sep>
gxps_images_create_from_png (GXPSArchive *zip,
                       const gchar *image_uri,
                       GError     **error)
{
#ifdef HAVE_LIBPNG
      GInputStream  *stream;
      GXPSImage     *image = NULL;
      char          *png_err_msg = NULL;
      png_struct    *png;
      png_info      *info;
      png_byte      *data = NULL;
      png_byte     **row_pointers = NULL;
      png_uint_32    png_width, png_height;
      int            depth, color_type, interlace, stride;
      unsigned int   i;
      cairo_format_t format;
      cairo_status_t status;

      stream = gxps_archive_open (zip, image_uri);
      if (!stream) {
            g_set_error (error,
                       GXPS_ERROR,
                       GXPS_ERROR_SOURCE_NOT_FOUND,
                       "Image source %s not found in archive",
                       image_uri);
            return NULL;
      }

      png = png_create_read_struct (PNG_LIBPNG_VER_STRING,
                              &png_err_msg,
                              png_error_callback,
                              png_warning_callback);
      if (png == NULL) {
            fill_png_error (error, image_uri, NULL);
```

```
      g_object_unref (stream);
      return NULL;
}

info = png_create_info_struct (png);
if (info == NULL) {
      fill_png_error (error, image_uri, NULL);
      g_object_unref (stream);
      png_destroy_read_struct (&png, NULL, NULL);
      return NULL;
}

png_set_read_fn (png, stream, _read_png);

if (setjmp (png_jmpbuf (png))) {
      fill_png_error (error, image_uri, png_err_msg);
      g_free (png_err_msg);
      g_object_unref (stream);
      png_destroy_read_struct (&png, &info, NULL);
      gxps_image_free (image);
      g_free (row_pointers);
      g_free (data);
      return NULL;
}

png_read_info (png, info);

png_get_IHDR (png, info,
            &png_width, &png_height, &depth,
            &color_type, &interlace, NULL, NULL);

/* convert palette/gray image to rgb */
if (color_type == PNG_COLOR_TYPE_PALETTE)
      png_set_palette_to_rgb (png);

/* expand gray bit depth if needed */
if (color_type == PNG_COLOR_TYPE_GRAY)
      png_set_expand_gray_1_2_4_to_8 (png);

/* transform transparency to alpha */
if (png_get_valid (png, info, PNG_INFO_tRNS))
      png_set_tRNS_to_alpha (png);

if (depth == 16)
      png_set_strip_16 (png);

if (depth < 8)
      png_set_packing (png);

/* convert grayscale to RGB */
if (color_type == PNG_COLOR_TYPE_GRAY ||
    color_type == PNG_COLOR_TYPE_GRAY_ALPHA)
      png_set_gray_to_rgb (png);
```

```
      if (interlace != PNG_INTERLACE_NONE)
            png_set_interlace_handling (png);

      png_set_filler (png, 0xff, PNG_FILLER_AFTER);

      /* recheck header after setting EXPAND options */
      png_read_update_info (png, info);
      png_get_IHDR (png, info,
                  &png_width, &png_height, &depth,
                  &color_type, &interlace, NULL, NULL);
      if (depth != 8 ||
          !(color_type == PNG_COLOR_TYPE_RGB ||
              color_type == PNG_COLOR_TYPE_RGB_ALPHA)) {
            fill_png_error (error, image_uri, NULL);
            g_object_unref (stream);
            png_destroy_read_struct (&png, &info, NULL);
            return NULL;
      }

      switch (color_type) {
      default:
            g_assert_not_reached();
            /* fall-through just in case ;-) */

      case PNG_COLOR_TYPE_RGB_ALPHA:
            format = CAIRO_FORMAT_ARGB32;
            png_set_read_user_transform_fn (png, premultiply_data);
            break;

      case PNG_COLOR_TYPE_RGB:
            format = CAIRO_FORMAT_RGB24;
            png_set_read_user_transform_fn (png, convert_bytes_to_data);
            break;
      }

      stride = cairo_format_stride_for_width (format, png_width);
      if (stride < 0) {
            fill_png_error (error, image_uri, NULL);
            g_object_unref (stream);
            png_destroy_read_struct (&png, &info, NULL);
            return NULL;
      }

      image = g_slice_new0 (GXPSImage);
      image->res_x = png_get_x_pixels_per_meter (png, info) *
METERS_PER_INCH;
      if (image->res_x == 0)
            image->res_x = 96;
      image->res_y = png_get_y_pixels_per_meter (png, info) *
METERS_PER_INCH;
      if (image->res_y == 0)
            image->res_y = 96;

      data = g_malloc (png_height * stride);
```

```
      row_pointers = g_new (png_byte *, png_height);

      for (i = 0; i < png_height; i++)
            row_pointers[i] = &data[i * stride];

      png_read_image (png, row_pointers);
      png_read_end (png, info);
      png_destroy_read_struct (&png, &info, NULL);
      g_object_unref (stream);
      g_free (row_pointers);

      image->surface = cairo_image_surface_create_for_data (data, format,
                                                  png_width, png_height,
                                                  stride);
      if (cairo_surface_status (image->surface)) {
            fill_png_error (error, image_uri, NULL);
            gxps_image_free (image);
            g_free (data);
            return NULL;
      }

      status = cairo_surface_set_user_data (image->surface,
                                  &image_data_cairo_key,
                                  data,
                                  (cairo_destroy_func_t) g_free);
      if (status) {
            fill_png_error (error, image_uri, NULL);
            gxps_image_free (image);
            g_free (data);
            return NULL;
      }

      return image;
#else
    return NULL;
#endif  /* HAVE_LIBPNG */
}
<sep>
ip6t_do_table(struct sk_buff *skb,
            const struct nf_hook_state *state,
            struct xt_table *table)
{
      unsigned int hook = state->hook;
      static const char nulldevname[IFNAMSIZ]
__attribute__((aligned(sizeof(long))));
      /* Initializing verdict to NF_DROP keeps gcc happy. */
      unsigned int verdict = NF_DROP;
      const char *indev, *outdev;
      const void *table_base;
      struct ip6t_entry *e, **jumpstack;
      unsigned int stackidx, cpu;
      const struct xt_table_info *private;
      struct xt_action_param acpar;
      unsigned int addend;
```

```
	/* Initialization */
	stackidx = 0;
	indev = state->in ? state->in->name : nulldevname;
	outdev = state->out ? state->out->name : nulldevname;
	/* We handle fragments by dealing with the first fragment as
	 * if it was a normal packet.  All other fragments are treated
	 * normally, except that they will NEVER match rules that ask
	 * things we don't know, ie. tcp syn flag or ports).  If the
	 * rule is also a fragment-specific rule, non-fragments won't
	 * match it. */
	acpar.hotdrop = false;
	acpar.state   = state;

	WARN_ON(!(table->valid_hooks & (1 << hook)));

	local_bh_disable();
	addend = xt_write_recseq_begin();
	private = READ_ONCE(table->private); /* Address dependency. */
	cpu        = smp_processor_id();
	table_base = private->entries;
	jumpstack  = (struct ip6t_entry **)private->jumpstack[cpu];

	/* Switch to alternate jumpstack if we're being invoked via TEE.
	 * TEE issues XT_CONTINUE verdict on original skb so we must not
	 * clobber the jumpstack.
	 *
	 * For recursion via REJECT or SYNPROXY the stack will be clobbered
	 * but it is no problem since absolute verdict is issued by these.
	 */
	if (static_key_false(&xt_tee_enabled))
		jumpstack += private->stacksize *
__this_cpu_read(nf_skb_duplicated);

	e = get_entry(table_base, private->hook_entry[hook]);

	do {
		const struct xt_entry_target *t;
		const struct xt_entry_match *ematch;
		struct xt_counters *counter;

		WARN_ON(!e);
		acpar.thoff = 0;
		if (!ip6_packet_match(skb, indev, outdev, &e->ipv6,
		    &acpar.thoff, &acpar.fragoff, &acpar.hotdrop)) {
 no_match:
			e = ip6t_next_entry(e);
			continue;
		}

		xt_ematch_foreach(ematch, e) {
			acpar.match     = ematch->u.kernel.match;
			acpar.matchinfo = ematch->data;
			if (!acpar.match->match(skb, &acpar))
```

```
                        goto no_match;
                }

                counter = xt_get_this_cpu_counter(&e->counters);
                ADD_COUNTER(*counter, skb->len, 1);

                t = ip6t_get_target_c(e);
                WARN_ON(!t->u.kernel.target);

#if IS_ENABLED(CONFIG_NETFILTER_XT_TARGET_TRACE)
                /* The packet is traced: log it */
                if (unlikely(skb->nf_trace))
                        trace_packet(state->net, skb, hook, state->in,
                                     state->out, table->name, private, e);
#endif
                /* Standard target? */
                if (!t->u.kernel.target->target) {
                        int v;

                        v = ((struct xt_standard_target *)t)->verdict;
                        if (v < 0) {
                                /* Pop from stack? */
                                if (v != XT_RETURN) {
                                        verdict = (unsigned int)(-v) - 1;
                                        break;
                                }
                                if (stackidx == 0)
                                        e = get_entry(table_base,
                                            private->underflow[hook]);
                                else
                                        e = ip6t_next_entry(jumpstack[--stackidx]);
                                continue;
                        }
                        if (table_base + v != ip6t_next_entry(e) &&
                            !(e->ipv6.flags & IP6T_F_GOTO)) {
                                jumpstack[stackidx++] = e;
                        }

                        e = get_entry(table_base, v);
                        continue;
                }

                acpar.target   = t->u.kernel.target;
                acpar.targinfo = t->data;

                verdict = t->u.kernel.target->target(skb, &acpar);
                if (verdict == XT_CONTINUE)
                        e = ip6t_next_entry(e);
                else
                        /* Verdict */
                        break;
        } while (!acpar.hotdrop);

        xt_write_recseq_end(addend);
```

```
      local_bh_enable();

      if (acpar.hotdrop)
            return NF_DROP;
      else return verdict;
}
<sep>
file_asynch_zero (struct rw *rw, struct command *command,
                  nbd_completion_callback cb, bool allocate)
{
  int dummy = 0;

  if (!file_synch_zero (rw, command->offset, command->slice.len,
allocate))
    return false;
  if (cb.callback (cb.user_data, &dummy) == -1) {
    perror (rw->name);
    exit (EXIT_FAILURE);
  }
  return true;
}
<sep>
uint8_t* FAST_FUNC udhcp_get_option32(struct dhcp_packet *packet, int
code)
{
      uint8_t *r = udhcp_get_option(packet, code);
      if (r) {
            if (r[-1] != 4)
                  r = NULL;
      }
      return r;
}
<sep>
ossl_cipher_initialize(VALUE self, VALUE str)
{
    EVP_CIPHER_CTX *ctx;
    const EVP_CIPHER *cipher;
    char *name;
    unsigned char dummy_key[EVP_MAX_KEY_LENGTH] = { 0 };

    name = StringValueCStr(str);
    GetCipherInit(self, ctx);
    if (ctx) {
      ossl_raise(rb_eRuntimeError, "Cipher already inititalized!");
    }
    AllocCipher(self, ctx);
    if (!(cipher = EVP_get_cipherbyname(name))) {
      ossl_raise(rb_eRuntimeError, "unsupported cipher algorithm
(%"PRIsVALUE")", str);
    }
    /*
     * EVP_CipherInit_ex() allows to specify NULL to key and IV, however
some
     * ciphers don't handle well (OpenSSL's bug). [Bug #2768]
```

```
     *
     * The EVP which has EVP_CIPH_RAND_KEY flag (such as DES3) allows
     * uninitialized key, but other EVPs (such as AES) does not allow it.
     * Calling EVP_CipherUpdate() without initializing key causes SEGV so
we
     * set the data filled with "\0" as the key by default.
     */
    if (EVP_CipherInit_ex(ctx, cipher, NULL, dummy_key, NULL, -1) != 1)
      ossl_raise(eCipherError, NULL);

    return self;
}
<sep>
GST_START_TEST (test_GstDateTime_iso8601)
{
  GstDateTime *dt, *dt2;
  gchar *str, *str2;
  GDateTime *gdt, *gdt2;

  dt = gst_date_time_new_now_utc ();
  fail_unless (gst_date_time_has_year (dt));
  fail_unless (gst_date_time_has_month (dt));
  fail_unless (gst_date_time_has_day (dt));
  fail_unless (gst_date_time_has_time (dt));
  fail_unless (gst_date_time_has_second (dt));
  str = gst_date_time_to_iso8601_string (dt);
  fail_unless (str != NULL);
  fail_unless_equals_int (strlen (str), strlen ("2012-06-26T22:46:43Z"));
  fail_unless (g_str_has_suffix (str, "Z"));
  dt2 = gst_date_time_new_from_iso8601_string (str);
  fail_unless (gst_date_time_get_year (dt) == gst_date_time_get_year
(dt2));
  fail_unless (gst_date_time_get_month (dt) == gst_date_time_get_month
(dt2));
  fail_unless (gst_date_time_get_day (dt) == gst_date_time_get_day
(dt2));
  fail_unless (gst_date_time_get_hour (dt) == gst_date_time_get_hour
(dt2));
  fail_unless (gst_date_time_get_minute (dt) == gst_date_time_get_minute
(dt2));
  fail_unless (gst_date_time_get_second (dt) == gst_date_time_get_second
(dt2));
  /* This will succeed because we're not comparing microseconds when
   * checking for equality */
  fail_unless (date_times_are_equal (dt, dt2));
  str2 = gst_date_time_to_iso8601_string (dt2);
  fail_unless_equals_string (str, str2);
  g_free (str2);
  gst_date_time_unref (dt2);
  g_free (str);
  gst_date_time_unref (dt);

  /* ---- year only ---- */
```

```
  dt = gst_date_time_new_y (2010);
  fail_unless (gst_date_time_has_year (dt));
  fail_unless (!gst_date_time_has_month (dt));
  fail_unless (!gst_date_time_has_day (dt));
  fail_unless (!gst_date_time_has_time (dt));
  fail_unless (!gst_date_time_has_second (dt));
  str = gst_date_time_to_iso8601_string (dt);
  fail_unless (str != NULL);
  fail_unless_equals_string (str, "2010");
  dt2 = gst_date_time_new_from_iso8601_string (str);
  fail_unless (gst_date_time_get_year (dt) == gst_date_time_get_year
(dt2));
  fail_unless (date_times_are_equal (dt, dt2));
  str2 = gst_date_time_to_iso8601_string (dt2);
  fail_unless_equals_string (str, str2);
  g_free (str2);
  gst_date_time_unref (dt2);
  g_free (str);
  gst_date_time_unref (dt);

  /* ---- year and month ---- */

  dt = gst_date_time_new_ym (2010, 10);
  fail_unless (gst_date_time_has_year (dt));
  fail_unless (gst_date_time_has_month (dt));
  fail_unless (!gst_date_time_has_day (dt));
  fail_unless (!gst_date_time_has_time (dt));
  fail_unless (!gst_date_time_has_second (dt));
  str = gst_date_time_to_iso8601_string (dt);
  fail_unless (str != NULL);
  fail_unless_equals_string (str, "2010-10");
  dt2 = gst_date_time_new_from_iso8601_string (str);
  fail_unless (gst_date_time_get_year (dt) == gst_date_time_get_year
(dt2));
  fail_unless (gst_date_time_get_month (dt) == gst_date_time_get_month
(dt2));
  fail_unless (date_times_are_equal (dt, dt2));
  str2 = gst_date_time_to_iso8601_string (dt2);
  fail_unless_equals_string (str, str2);
  g_free (str2);
  gst_date_time_unref (dt2);
  g_free (str);
  gst_date_time_unref (dt);

  /* ---- year and month ---- */

  dt = gst_date_time_new_ymd (2010, 10, 30);
  fail_unless (gst_date_time_has_year (dt));
  fail_unless (gst_date_time_has_month (dt));
  fail_unless (gst_date_time_has_day (dt));
  fail_unless (!gst_date_time_has_time (dt));
  fail_unless (!gst_date_time_has_second (dt));
  str = gst_date_time_to_iso8601_string (dt);
  fail_unless (str != NULL);
```

```
  fail_unless_equals_string (str, "2010-10-30");
  dt2 = gst_date_time_new_from_iso8601_string (str);
  fail_unless (gst_date_time_get_year (dt) == gst_date_time_get_year
(dt2));
  fail_unless (gst_date_time_get_month (dt) == gst_date_time_get_month
(dt2));
  fail_unless (gst_date_time_get_day (dt) == gst_date_time_get_day
(dt2));
  fail_unless (date_times_are_equal (dt, dt2));
  str2 = gst_date_time_to_iso8601_string (dt2);
  fail_unless_equals_string (str, str2);
  g_free (str2);
  gst_date_time_unref (dt2);
  g_free (str);
  gst_date_time_unref (dt);

  /* ---- date and time, but no seconds ---- */

  dt = gst_date_time_new (-4.5, 2010, 10, 30, 15, 50, -1);
  fail_unless (gst_date_time_has_year (dt));
  fail_unless (gst_date_time_has_month (dt));
  fail_unless (gst_date_time_has_day (dt));
  fail_unless (gst_date_time_has_time (dt));
  fail_unless (!gst_date_time_has_second (dt));
  str = gst_date_time_to_iso8601_string (dt);
  fail_unless (str != NULL);
  fail_unless_equals_string (str, "2010-10-30T15:50-0430");
  dt2 = gst_date_time_new_from_iso8601_string (str);
  fail_unless (gst_date_time_get_year (dt) == gst_date_time_get_year
(dt2));
  fail_unless (gst_date_time_get_month (dt) == gst_date_time_get_month
(dt2));
  fail_unless (gst_date_time_get_day (dt) == gst_date_time_get_day
(dt2));
  fail_unless (gst_date_time_get_hour (dt) == gst_date_time_get_hour
(dt2));
  fail_unless (gst_date_time_get_minute (dt) == gst_date_time_get_minute
(dt2));
  fail_unless (date_times_are_equal (dt, dt2));
  str2 = gst_date_time_to_iso8601_string (dt2);
  fail_unless_equals_string (str, str2);
  g_free (str2);
  gst_date_time_unref (dt2);
  g_free (str);
  gst_date_time_unref (dt);

  /* ---- date and time, but no seconds (UTC) ---- */

  dt = gst_date_time_new (0, 2010, 10, 30, 15, 50, -1);
  fail_unless (gst_date_time_has_year (dt));
  fail_unless (gst_date_time_has_month (dt));
  fail_unless (gst_date_time_has_day (dt));
  fail_unless (gst_date_time_has_time (dt));
  fail_unless (!gst_date_time_has_second (dt));
```

```
  str = gst_date_time_to_iso8601_string (dt);
  fail_unless (str != NULL);
  fail_unless_equals_string (str, "2010-10-30T15:50Z");
  dt2 = gst_date_time_new_from_iso8601_string (str);
  fail_unless (gst_date_time_get_year (dt) == gst_date_time_get_year
(dt2));
  fail_unless (gst_date_time_get_month (dt) == gst_date_time_get_month
(dt2));
  fail_unless (gst_date_time_get_day (dt) == gst_date_time_get_day
(dt2));
  fail_unless (gst_date_time_get_hour (dt) == gst_date_time_get_hour
(dt2));
  fail_unless (gst_date_time_get_minute (dt) == gst_date_time_get_minute
(dt2));
  fail_unless (date_times_are_equal (dt, dt2));
  str2 = gst_date_time_to_iso8601_string (dt2);
  fail_unless_equals_string (str, str2);
  g_free (str2);
  gst_date_time_unref (dt2);
  g_free (str);
  gst_date_time_unref (dt);

  /* ---- date and time, with seconds ---- */

  dt = gst_date_time_new (-4.5, 2010, 10, 30, 15, 50, 0);
  fail_unless (gst_date_time_has_year (dt));
  fail_unless (gst_date_time_has_month (dt));
  fail_unless (gst_date_time_has_day (dt));
  fail_unless (gst_date_time_has_time (dt));
  fail_unless (gst_date_time_has_second (dt));
  str = gst_date_time_to_iso8601_string (dt);
  fail_unless (str != NULL);
  fail_unless_equals_string (str, "2010-10-30T15:50:00-0430");
  dt2 = gst_date_time_new_from_iso8601_string (str);
  fail_unless (gst_date_time_get_year (dt) == gst_date_time_get_year
(dt2));
  fail_unless (gst_date_time_get_month (dt) == gst_date_time_get_month
(dt2));
  fail_unless (gst_date_time_get_day (dt) == gst_date_time_get_day
(dt2));
  fail_unless (gst_date_time_get_hour (dt) == gst_date_time_get_hour
(dt2));
  fail_unless (gst_date_time_get_minute (dt) == gst_date_time_get_minute
(dt2));
  fail_unless (date_times_are_equal (dt, dt2));
  str2 = gst_date_time_to_iso8601_string (dt2);
  fail_unless_equals_string (str, str2);
  g_free (str2);
  gst_date_time_unref (dt2);
  g_free (str);
  gst_date_time_unref (dt);

  /* ---- date and time, with seconds (UTC) ---- */
```

```
  dt = gst_date_time_new (0, 2010, 10, 30, 15, 50, 0);
  fail_unless (gst_date_time_has_year (dt));
  fail_unless (gst_date_time_has_month (dt));
  fail_unless (gst_date_time_has_day (dt));
  fail_unless (gst_date_time_has_time (dt));
  fail_unless (gst_date_time_has_second (dt));
  str = gst_date_time_to_iso8601_string (dt);
  fail_unless (str != NULL);
  fail_unless_equals_string (str, "2010-10-30T15:50:00Z");
  dt2 = gst_date_time_new_from_iso8601_string (str);
  fail_unless (gst_date_time_get_year (dt) == gst_date_time_get_year
(dt2));
  fail_unless (gst_date_time_get_month (dt) == gst_date_time_get_month
(dt2));
  fail_unless (gst_date_time_get_day (dt) == gst_date_time_get_day
(dt2));
  fail_unless (gst_date_time_get_hour (dt) == gst_date_time_get_hour
(dt2));
  fail_unless (gst_date_time_get_minute (dt) == gst_date_time_get_minute
(dt2));
  fail_unless (date_times_are_equal (dt, dt2));
  str2 = gst_date_time_to_iso8601_string (dt2);
  fail_unless_equals_string (str, str2);
  g_free (str2);
  gst_date_time_unref (dt2);
  g_free (str);
  gst_date_time_unref (dt);

  /* ---- date and time, but without the 'T' and without timezone */
  dt = gst_date_time_new_from_iso8601_string ("2010-10-30 15:50");
  fail_unless (gst_date_time_get_year (dt) == 2010);
  fail_unless (gst_date_time_get_month (dt) == 10);
  fail_unless (gst_date_time_get_day (dt) == 30);
  fail_unless (gst_date_time_get_hour (dt) == 15);
  fail_unless (gst_date_time_get_minute (dt) == 50);
  fail_unless (!gst_date_time_has_second (dt));
  gst_date_time_unref (dt);

  /* ---- date and time+secs, but without the 'T' and without timezone */
  dt = gst_date_time_new_from_iso8601_string ("2010-10-30 15:50:33");
  fail_unless (gst_date_time_get_year (dt) == 2010);
  fail_unless (gst_date_time_get_month (dt) == 10);
  fail_unless (gst_date_time_get_day (dt) == 30);
  fail_unless (gst_date_time_get_hour (dt) == 15);
  fail_unless (gst_date_time_get_minute (dt) == 50);
  fail_unless (gst_date_time_get_second (dt) == 33);
  gst_date_time_unref (dt);

  /* ---- dates with 00s */
  dt = gst_date_time_new_from_iso8601_string ("2010-10-00");
  fail_unless (gst_date_time_get_year (dt) == 2010);
  fail_unless (gst_date_time_get_month (dt) == 10);
  fail_unless (!gst_date_time_has_day (dt));
  fail_unless (!gst_date_time_has_time (dt));
```

```
  gst_date_time_unref (dt);

  dt = gst_date_time_new_from_iso8601_string ("2010-00-00");
  fail_unless (gst_date_time_get_year (dt) == 2010);
  fail_unless (!gst_date_time_has_month (dt));
  fail_unless (!gst_date_time_has_day (dt));
  fail_unless (!gst_date_time_has_time (dt));
  gst_date_time_unref (dt);

  dt = gst_date_time_new_from_iso8601_string ("2010-00-30");
  fail_unless (gst_date_time_get_year (dt) == 2010);
  fail_unless (!gst_date_time_has_month (dt));
  fail_unless (!gst_date_time_has_day (dt));
  fail_unless (!gst_date_time_has_time (dt));
  gst_date_time_unref (dt);

  /* completely invalid */
  dt = gst_date_time_new_from_iso8601_string ("0000-00-00");
  fail_unless (dt == NULL);

  /* partially invalid - here we'll just extract the year */
  dt = gst_date_time_new_from_iso8601_string ("2010/05/30");
  fail_unless (gst_date_time_get_year (dt) == 2010);
  fail_unless (!gst_date_time_has_month (dt));
  fail_unless (!gst_date_time_has_day (dt));
  fail_unless (!gst_date_time_has_time (dt));
  gst_date_time_unref (dt);


  /* only time provided - we assume today's date */
  gdt = g_date_time_new_now_utc ();

  dt = gst_date_time_new_from_iso8601_string ("15:50:33");
  fail_unless (gst_date_time_get_year (dt) == g_date_time_get_year
(gdt));
  fail_unless (gst_date_time_get_month (dt) == g_date_time_get_month
(gdt));
  fail_unless (gst_date_time_get_day (dt) ==
      g_date_time_get_day_of_month (gdt));
  fail_unless (gst_date_time_get_hour (dt) == 15);
  fail_unless (gst_date_time_get_minute (dt) == 50);
  fail_unless (gst_date_time_get_second (dt) == 33);
  gst_date_time_unref (dt);

  dt = gst_date_time_new_from_iso8601_string ("15:50:33Z");
  fail_unless (gst_date_time_get_year (dt) == g_date_time_get_year
(gdt));
  fail_unless (gst_date_time_get_month (dt) == g_date_time_get_month
(gdt));
  fail_unless (gst_date_time_get_day (dt) ==
      g_date_time_get_day_of_month (gdt));
  fail_unless (gst_date_time_get_hour (dt) == 15);
  fail_unless (gst_date_time_get_minute (dt) == 50);
  fail_unless (gst_date_time_get_second (dt) == 33);
```

```
  gst_date_time_unref (dt);

  dt = gst_date_time_new_from_iso8601_string ("15:50");
  fail_unless (gst_date_time_get_year (dt) == g_date_time_get_year
(gdt));
  fail_unless (gst_date_time_get_month (dt) == g_date_time_get_month
(gdt));
  fail_unless (gst_date_time_get_day (dt) ==
      g_date_time_get_day_of_month (gdt));
  fail_unless (gst_date_time_get_hour (dt) == 15);
  fail_unless (gst_date_time_get_minute (dt) == 50);
  fail_unless (!gst_date_time_has_second (dt));
  gst_date_time_unref (dt);

  dt = gst_date_time_new_from_iso8601_string ("15:50Z");
  fail_unless (gst_date_time_get_year (dt) == g_date_time_get_year
(gdt));
  fail_unless (gst_date_time_get_month (dt) == g_date_time_get_month
(gdt));
  fail_unless (gst_date_time_get_day (dt) ==
      g_date_time_get_day_of_month (gdt));
  fail_unless (gst_date_time_get_hour (dt) == 15);
  fail_unless (gst_date_time_get_minute (dt) == 50);
  fail_unless (!gst_date_time_has_second (dt));
  gst_date_time_unref (dt);

  gdt2 = g_date_time_add_minutes (gdt, -270);
  g_date_time_unref (gdt);

  dt = gst_date_time_new_from_iso8601_string ("15:50:33-0430");
  fail_unless (gst_date_time_get_year (dt) == g_date_time_get_year
(gdt2));
  fail_unless (gst_date_time_get_month (dt) == g_date_time_get_month
(gdt2));
  fail_unless (gst_date_time_get_day (dt) ==
      g_date_time_get_day_of_month (gdt2));
  fail_unless (gst_date_time_get_hour (dt) == 15);
  fail_unless (gst_date_time_get_minute (dt) == 50);
  fail_unless (gst_date_time_get_second (dt) == 33);
  gst_date_time_unref (dt);

  dt = gst_date_time_new_from_iso8601_string ("15:50-0430");
  fail_unless (gst_date_time_get_year (dt) == g_date_time_get_year
(gdt2));
  fail_unless (gst_date_time_get_month (dt) == g_date_time_get_month
(gdt2));
  fail_unless (gst_date_time_get_day (dt) ==
      g_date_time_get_day_of_month (gdt2));
  fail_unless (gst_date_time_get_hour (dt) == 15);
  fail_unless (gst_date_time_get_minute (dt) == 50);
  fail_unless (!gst_date_time_has_second (dt));
  gst_date_time_unref (dt);

  g_date_time_unref (gdt2);
```

```
}
<sep>
dwg_free_object (Dwg_Object *obj)
{
  int error = 0;
  long unsigned int j;
  Dwg_Data *dwg;
  Bit_Chain *dat = &pdat;

  if (obj && obj->parent)
    {
      dwg = obj->parent;
      dat->version = dwg->header.version;
    }
  else
    return;
  if (obj->type == DWG_TYPE_FREED || obj->tio.object == NULL)
    return;
  dat->from_version = dat->version;
  if (obj->supertype == DWG_SUPERTYPE_UNKNOWN)
    goto unhandled;

  switch (obj->type)
    {
    case DWG_TYPE_TEXT:
      dwg_free_TEXT (dat, obj);
      break;
    case DWG_TYPE_ATTRIB:
      dwg_free_ATTRIB (dat, obj);
      break;
    case DWG_TYPE_ATTDEF:
      dwg_free_ATTDEF (dat, obj);
      break;
    case DWG_TYPE_BLOCK:
      dwg_free_BLOCK (dat, obj);
      break;
    case DWG_TYPE_ENDBLK:
      dwg_free_ENDBLK (dat, obj);
      break;
    case DWG_TYPE_SEQEND:
      dwg_free_SEQEND (dat, obj);
      break;
    case DWG_TYPE_INSERT:
      dwg_free_INSERT (dat, obj);
      break;
    case DWG_TYPE_MINSERT:
      dwg_free_MINSERT (dat, obj);
      break;
    case DWG_TYPE_VERTEX_2D:
      dwg_free_VERTEX_2D (dat, obj);
      break;
    case DWG_TYPE_VERTEX_3D:
      dwg_free_VERTEX_3D (dat, obj);
      break;
```

```
case DWG_TYPE_VERTEX_MESH:
  dwg_free_VERTEX_MESH (dat, obj);
  break;
case DWG_TYPE_VERTEX_PFACE:
  dwg_free_VERTEX_PFACE (dat, obj);
  break;
case DWG_TYPE_VERTEX_PFACE_FACE:
  dwg_free_VERTEX_PFACE_FACE (dat, obj);
  break;
case DWG_TYPE_POLYLINE_2D:
  dwg_free_POLYLINE_2D (dat, obj);
  break;
case DWG_TYPE_POLYLINE_3D:
  dwg_free_POLYLINE_3D (dat, obj);
  break;
case DWG_TYPE_ARC:
  dwg_free_ARC (dat, obj);
  break;
case DWG_TYPE_CIRCLE:
  dwg_free_CIRCLE (dat, obj);
  break;
case DWG_TYPE_LINE:
  dwg_free_LINE (dat, obj);
  break;
case DWG_TYPE_DIMENSION_ORDINATE:
  dwg_free_DIMENSION_ORDINATE (dat, obj);
  break;
case DWG_TYPE_DIMENSION_LINEAR:
  dwg_free_DIMENSION_LINEAR (dat, obj);
  break;
case DWG_TYPE_DIMENSION_ALIGNED:
  dwg_free_DIMENSION_ALIGNED (dat, obj);
  break;
case DWG_TYPE_DIMENSION_ANG3PT:
  dwg_free_DIMENSION_ANG3PT (dat, obj);
  break;
case DWG_TYPE_DIMENSION_ANG2LN:
  dwg_free_DIMENSION_ANG2LN (dat, obj);
  break;
case DWG_TYPE_DIMENSION_RADIUS:
  dwg_free_DIMENSION_RADIUS (dat, obj);
  break;
case DWG_TYPE_DIMENSION_DIAMETER:
  dwg_free_DIMENSION_DIAMETER (dat, obj);
  break;
case DWG_TYPE_POINT:
  dwg_free_POINT (dat, obj);
  break;
case DWG_TYPE__3DFACE:
  dwg_free__3DFACE (dat, obj);
  break;
case DWG_TYPE_POLYLINE_PFACE:
  dwg_free_POLYLINE_PFACE (dat, obj);
  break;
```

```
case DWG_TYPE_POLYLINE_MESH:
  dwg_free_POLYLINE_MESH (dat, obj);
  break;
case DWG_TYPE_SOLID:
  dwg_free_SOLID (dat, obj);
  break;
case DWG_TYPE_TRACE:
  dwg_free_TRACE (dat, obj);
  break;
case DWG_TYPE_SHAPE:
  dwg_free_SHAPE (dat, obj);
  break;
case DWG_TYPE_VIEWPORT:
  dwg_free_VIEWPORT (dat, obj);
  break;
case DWG_TYPE_ELLIPSE:
  dwg_free_ELLIPSE (dat, obj);
  break;
case DWG_TYPE_SPLINE:
  dwg_free_SPLINE (dat, obj);
  break;
case DWG_TYPE_REGION:
  dwg_free_REGION (dat, obj);
  break;
case DWG_TYPE__3DSOLID:
  dwg_free__3DSOLID (dat, obj);
  break; /* Check the type of the object */
case DWG_TYPE_BODY:
  dwg_free_BODY (dat, obj);
  break;
case DWG_TYPE_RAY:
  dwg_free_RAY (dat, obj);
  break;
case DWG_TYPE_XLINE:
  dwg_free_XLINE (dat, obj);
  break;
case DWG_TYPE_DICTIONARY:
  dwg_free_DICTIONARY (dat, obj);
  break;
case DWG_TYPE_MTEXT:
  dwg_free_MTEXT (dat, obj);
  break;
case DWG_TYPE_LEADER:
  dwg_free_LEADER (dat, obj);
  break;
case DWG_TYPE_TOLERANCE:
  dwg_free_TOLERANCE (dat, obj);
  break;
case DWG_TYPE_MLINE:
  dwg_free_MLINE (dat, obj);
  break;
case DWG_TYPE_BLOCK_CONTROL:
  dwg_free_BLOCK_CONTROL (dat, obj);
  break;
```

```
case DWG_TYPE_BLOCK_HEADER:
  dwg_free_BLOCK_HEADER (dat, obj);
  break;
case DWG_TYPE_LAYER_CONTROL:
  dwg_free_LAYER_CONTROL (dat, obj);
  break;
case DWG_TYPE_LAYER:
  dwg_free_LAYER (dat, obj);
  break;
case DWG_TYPE_STYLE_CONTROL:
  dwg_free_STYLE_CONTROL (dat, obj);
  break;
case DWG_TYPE_STYLE:
  dwg_free_STYLE (dat, obj);
  break;
case DWG_TYPE_LTYPE_CONTROL:
  dwg_free_LTYPE_CONTROL (dat, obj);
  break;
case DWG_TYPE_LTYPE:
  dwg_free_LTYPE (dat, obj);
  break;
case DWG_TYPE_VIEW_CONTROL:
  dwg_free_VIEW_CONTROL (dat, obj);
  break;
case DWG_TYPE_VIEW:
  dwg_free_VIEW (dat, obj);
  break;
case DWG_TYPE_UCS_CONTROL:
  dwg_free_UCS_CONTROL (dat, obj);
  break;
case DWG_TYPE_UCS:
  dwg_free_UCS (dat, obj);
  break;
case DWG_TYPE_VPORT_CONTROL:
  dwg_free_VPORT_CONTROL (dat, obj);
  break;
case DWG_TYPE_VPORT:
  dwg_free_VPORT (dat, obj);
  break;
case DWG_TYPE_APPID_CONTROL:
  dwg_free_APPID_CONTROL (dat, obj);
  break;
case DWG_TYPE_APPID:
  dwg_free_APPID (dat, obj);
  break;
case DWG_TYPE_DIMSTYLE_CONTROL:
  dwg_free_DIMSTYLE_CONTROL (dat, obj);
  break;
case DWG_TYPE_DIMSTYLE:
  dwg_free_DIMSTYLE (dat, obj);
  break;
case DWG_TYPE_VPORT_ENTITY_CONTROL:
  dwg_free_VPORT_ENTITY_CONTROL (dat, obj);
  break;
```

```
    case DWG_TYPE_VPORT_ENTITY_HEADER:
      dwg_free_VPORT_ENTITY_HEADER (dat, obj);
      break;
    case DWG_TYPE_GROUP:
      dwg_free_GROUP (dat, obj);
      break;
    case DWG_TYPE_MLINESTYLE:
      dwg_free_MLINESTYLE (dat, obj);
      break;
    case DWG_TYPE_OLE2FRAME:
      dwg_free_OLE2FRAME (dat, obj);
      break;
    case DWG_TYPE_DUMMY:
      dwg_free_DUMMY (dat, obj);
      break;
    case DWG_TYPE_LONG_TRANSACTION:
      dwg_free_LONG_TRANSACTION (dat, obj);
      break;
    case DWG_TYPE_LWPOLYLINE:
      dwg_free_LWPOLYLINE (dat, obj);
      break;
    case DWG_TYPE_HATCH:
      dwg_free_HATCH (dat, obj);
      break;
    case DWG_TYPE_XRECORD:
      dwg_free_XRECORD (dat, obj);
      break;
    case DWG_TYPE_PLACEHOLDER:
      dwg_free_PLACEHOLDER (dat, obj);
      break;
    case DWG_TYPE_OLEFRAME:
      dwg_free_OLEFRAME (dat, obj);
      break;
#ifdef DEBUG_VBA_PROJECT
    case DWG_TYPE_VBA_PROJECT:
      dwg_free_VBA_PROJECT (dat, obj);
      break;
#endif
    case DWG_TYPE_LAYOUT:
      dwg_free_LAYOUT (dat, obj);
      break;
    case DWG_TYPE_PROXY_ENTITY:
      dwg_free_PROXY_ENTITY (dat, obj);
      break;
    case DWG_TYPE_PROXY_OBJECT:
      dwg_free_PROXY_OBJECT (dat, obj);
      break;
    default:
      if (obj->type == obj->parent->layout_type)
        {
          SINCE (R_13)
          {
            dwg_free_LAYOUT (dat, obj); // XXX avoid double-free, esp. in
eed
```

```
          }
        }

      else if ((error = dwg_free_variable_type (obj->parent, obj))
               & DWG_ERR_UNHANDLEDCLASS)
        {
          int is_entity;
          int i;
          Dwg_Class *klass;

        unhandled:
          is_entity = 0;
          i = obj->type - 500;
          klass = NULL;

          dwg = obj->parent;
          if (dwg->dwg_class && i >= 0 && i < (int)dwg->num_classes)
            {
              klass = &dwg->dwg_class[i];
              is_entity = klass ? dwg_class_is_entity (klass) : 0;
            }
          // indxf (and later injson) already creates some DEBUGGING
classes
          if (obj->fixedtype == DWG_TYPE_TABLE)
            {
              // just the preview, i.e. common. plus some colors: leak
              dwg_free_UNKNOWN_ENT (dat, obj);
            }
          else if (obj->fixedtype == DWG_TYPE_DATATABLE)
            {
              dwg_free_UNKNOWN_OBJ (dat, obj);
            }
          else if (klass && !is_entity)
            {
              dwg_free_UNKNOWN_OBJ (dat, obj);
            }
          else if (klass && is_entity)
            {
              dwg_free_UNKNOWN_ENT (dat, obj);
            }
          else // not a class
            {
              FREE_IF (obj->tio.unknown);
            }
        }
    }
  /* With this importer the dxfname is dynamic, just the name is const */
  if (dwg->opts & DWG_OPTS_INDXF)
    FREE_IF (obj->dxfname);
  obj->type = DWG_TYPE_FREED;
}
<sep>
asn1_der_decoding (asn1_node * element, const void *ider, int len,
                char *errorDescription)
```

```
{
  asn1_node node, p, p2, p3;
  char temp[128];
  int counter, len2, len3, len4, move, ris, tlen;
  unsigned char class;
  unsigned long tag;
  int indefinite, result;
  const unsigned char *der = ider;

  node = *element;

  if (errorDescription != NULL)
    errorDescription[0] = 0;

  if (node == NULL)
    return ASN1_ELEMENT_NOT_FOUND;

  if (node->type & CONST_OPTION)
    {
      result = ASN1_GENERIC_ERROR;
      goto cleanup;
    }

  counter = 0;
  move = DOWN;
  p = node;
  while (1)
    {
      ris = ASN1_SUCCESS;
      if (move != UP)
      {
        if (p->type & CONST_SET)
          {
            p2 = _asn1_find_up (p);
            len2 = _asn1_strtol (p2->value, NULL, 10);
            if (len2 == -1)
            {
              if (!der[counter] && !der[counter + 1])
                {
                  p = p2;
                  move = UP;
                  counter += 2;
                  continue;
                }
            }
            else if (counter == len2)
            {
              p = p2;
              move = UP;
              continue;
            }
            else if (counter > len2)
            {
              result = ASN1_DER_ERROR;
```

```
      goto cleanup;
    }
    p2 = p2->down;
    while (p2)
    {
      if ((p2->type & CONST_SET) && (p2->type & CONST_NOT_USED))
        {
          ris =
            extract_tag_der_recursive (p2, der + counter,
                                       len - counter, &len2);
          if (ris == ASN1_SUCCESS)
          {
            p2->type &= ~CONST_NOT_USED;
            p = p2;
            break;
          }
        }
      p2 = p2->right;
    }
    if (p2 == NULL)
    {
      result = ASN1_DER_ERROR;
      goto cleanup;
    }
  }

if ((p->type & CONST_OPTION) || (p->type & CONST_DEFAULT))
  {
    p2 = _asn1_find_up (p);
    len2 = _asn1_strtol (p2->value, NULL, 10);
    if (counter == len2)
    {
      if (p->right)
        {
          p2 = p->right;
          move = RIGHT;
        }
      else
        move = UP;

      if (p->type & CONST_OPTION)
        asn1_delete_structure (&p);

      p = p2;
      continue;
    }
  }

if (type_field (p->type) == ASN1_ETYPE_CHOICE)
  {
    while (p->down)
    {
      if (counter < len)
        ris =
```

```
                extract_tag_der_recursive (p->down, der + counter,
                                           len - counter, &len2);
              else
                ris = ASN1_DER_ERROR;
              if (ris == ASN1_SUCCESS)
                {
                  delete_unneeded_choice_fields(p->down);
                  break;
                }
              else if (ris == ASN1_ERROR_TYPE_ANY)
                {
                  result = ASN1_ERROR_TYPE_ANY;
                  goto cleanup;
                }
              else
                {
                  p2 = p->down;
                  asn1_delete_structure (&p2);
                }
            }

            if (p->down == NULL)
            {
              if (!(p->type & CONST_OPTION))
                {
                  result = ASN1_DER_ERROR;
                  goto cleanup;
                }
            }
            else if (type_field (p->type) != ASN1_ETYPE_CHOICE)
            p = p->down;
          }

        if ((p->type & CONST_OPTION) || (p->type & CONST_DEFAULT))
          {
            p2 = _asn1_find_up (p);
            len2 = _asn1_strtol (p2->value, NULL, 10);
            if ((len2 != -1) && (counter > len2))
            ris = ASN1_TAG_ERROR;
          }

        if (ris == ASN1_SUCCESS)
          ris =
            extract_tag_der_recursive (p, der + counter, len - counter,
&len2);
        if (ris != ASN1_SUCCESS)
          {
            if (p->type & CONST_OPTION)
            {
              p->type |= CONST_NOT_USED;
              move = RIGHT;
            }
            else if (p->type & CONST_DEFAULT)
            {
```

```
        _asn1_set_value (p, NULL, 0);
        move = RIGHT;
      }
      else
      {
        if (errorDescription != NULL)
          _asn1_error_description_tag_error (p, errorDescription);

        result = ASN1_TAG_ERROR;
        goto cleanup;
      }
    }
  else
    counter += len2;
}

if (ris == ASN1_SUCCESS)
{
  switch (type_field (p->type))
    {
    case ASN1_ETYPE_NULL:
      if (der[counter])
      {
        result = ASN1_DER_ERROR;
        goto cleanup;
      }
      counter++;
      move = RIGHT;
      break;
    case ASN1_ETYPE_BOOLEAN:
      if (der[counter++] != 1)
      {
        result = ASN1_DER_ERROR;
        goto cleanup;
      }
      if (der[counter++] == 0)
      _asn1_set_value (p, "F", 1);
      else
      _asn1_set_value (p, "T", 1);
      move = RIGHT;
      break;
    case ASN1_ETYPE_INTEGER:
    case ASN1_ETYPE_ENUMERATED:
      len2 =
      asn1_get_length_der (der + counter, len - counter, &len3);
      if (len2 < 0)
      {
        result = ASN1_DER_ERROR;
        goto cleanup;
      }

      _asn1_set_value (p, der + counter, len3 + len2);
      counter += len3 + len2;
      move = RIGHT;
```

```
            break;
          case ASN1_ETYPE_OBJECT_ID:
            result =
            _asn1_get_objectid_der (der + counter, len - counter, &len2,
                                    temp, sizeof (temp));
            if (result != ASN1_SUCCESS)
            goto cleanup;

            tlen = strlen (temp);
            if (tlen > 0)
            _asn1_set_value (p, temp, tlen + 1);
            counter += len2;
            move = RIGHT;
            break;
          case ASN1_ETYPE_GENERALIZED_TIME:
          case ASN1_ETYPE_UTC_TIME:
            result =
            _asn1_get_time_der (der + counter, len - counter, &len2,
temp,
                                sizeof (temp) - 1);
            if (result != ASN1_SUCCESS)
            goto cleanup;

            tlen = strlen (temp);
            if (tlen > 0)
            _asn1_set_value (p, temp, tlen);
            counter += len2;
            move = RIGHT;
            break;
          case ASN1_ETYPE_OCTET_STRING:
            len3 = len - counter;
            result = _asn1_get_octet_string (der + counter, p, &len3);
            if (result != ASN1_SUCCESS)
            goto cleanup;

            counter += len3;
            move = RIGHT;
            break;
          case ASN1_ETYPE_GENERALSTRING:
          case ASN1_ETYPE_NUMERIC_STRING:
          case ASN1_ETYPE_IA5_STRING:
          case ASN1_ETYPE_TELETEX_STRING:
          case ASN1_ETYPE_PRINTABLE_STRING:
          case ASN1_ETYPE_UNIVERSAL_STRING:
          case ASN1_ETYPE_BMP_STRING:
          case ASN1_ETYPE_UTF8_STRING:
          case ASN1_ETYPE_VISIBLE_STRING:
          case ASN1_ETYPE_BIT_STRING:
            len2 =
            asn1_get_length_der (der + counter, len - counter, &len3);
            if (len2 < 0)
            {
              result = ASN1_DER_ERROR;
              goto cleanup;
```

```
            }

            _asn1_set_value (p, der + counter, len3 + len2);
            counter += len3 + len2;
            move = RIGHT;
            break;
          case ASN1_ETYPE_SEQUENCE:
          case ASN1_ETYPE_SET:
            if (move == UP)
            {
              len2 = _asn1_strtol (p->value, NULL, 10);
              _asn1_set_value (p, NULL, 0);
              if (len2 == -1)
                {       /* indefinite length method */
                  if (len - counter + 1 > 0)
                  {
                    if ((der[counter]) || der[counter + 1])
                      {
                        result = ASN1_DER_ERROR;
                        goto cleanup;
                      }
                  }
                  else
                  {
                    result = ASN1_DER_ERROR;
                    goto cleanup;
                  }
                  counter += 2;
                }
              else
                {       /* definite length method */
                  if (len2 != counter)
                  {
                    result = ASN1_DER_ERROR;
                    goto cleanup;
                  }
                }
              move = RIGHT;
            }
            else
            {           /* move==DOWN || move==RIGHT */
              len3 =
                asn1_get_length_der (der + counter, len - counter,
&len2);
              if (len3 < -1)
                {
                  result = ASN1_DER_ERROR;
                  goto cleanup;
                }
              counter += len2;
              if (len3 > 0)
                {
                  _asn1_ltostr (counter + len3, temp);
                  tlen = strlen (temp);
```

```
        if (tlen > 0)
        _asn1_set_value (p, temp, tlen + 1);
        move = DOWN;
      }
    else if (len3 == 0)
      {
        p2 = p->down;
        while (p2)
        {
          if (type_field (p2->type) != ASN1_ETYPE_TAG)
            {
              p3 = p2->right;
              asn1_delete_structure (&p2);
              p2 = p3;
            }
          else
            p2 = p2->right;
        }
        move = RIGHT;
      }
    else
      {       /* indefinite length method */
        _asn1_set_value (p, "-1", 3);
        move = DOWN;
      }
  }
  break;
case ASN1_ETYPE_SEQUENCE_OF:
case ASN1_ETYPE_SET_OF:
  if (move == UP)
  {
    len2 = _asn1_strtol (p->value, NULL, 10);
    if (len2 == -1)
      {       /* indefinite length method */
        if ((counter + 2) > len)
        {
          result = ASN1_DER_ERROR;
          goto cleanup;
        }

        if ((der[counter]) || der[counter + 1])
        {
          _asn1_append_sequence_set (p);
          p = p->down;
          while (p->right)
            p = p->right;
          move = RIGHT;
          continue;
        }
        _asn1_set_value (p, NULL, 0);
        counter += 2;
      }
    else
      {       /* definite length method */
```

```
                if (len2 > counter)
                {
                  _asn1_append_sequence_set (p);
                  p = p->down;
                  while (p->right)
                    p = p->right;
                  move = RIGHT;
                  continue;
                }
                _asn1_set_value (p, NULL, 0);
                if (len2 != counter)
                {
                  result = ASN1_DER_ERROR;
                  goto cleanup;
                }
              }
            }
            else
            {           /* move==DOWN || move==RIGHT */
              len3 =
                asn1_get_length_der (der + counter, len - counter,
&len2);
              if (len3 < -1)
                {
                  result = ASN1_DER_ERROR;
                  goto cleanup;
                }
              counter += len2;
              if (len3)
                {
                  if (len3 > 0)
                  {     /* definite length method */
                    _asn1_ltostr (counter + len3, temp);
                    tlen = strlen (temp);

                    if (tlen > 0)
                      _asn1_set_value (p, temp, tlen + 1);
                  }
                  else
                  {     /* indefinite length method */
                    _asn1_set_value (p, "-1", 3);
                  }
                  p2 = p->down;
                  while ((type_field (p2->type) == ASN1_ETYPE_TAG)
                       || (type_field (p2->type) == ASN1_ETYPE_SIZE))
                    p2 = p2->right;
                  if (p2->right == NULL)
                    _asn1_append_sequence_set (p);
                  p = p2;
                }
            }
            move = RIGHT;
            break;
          case ASN1_ETYPE_ANY:
```

```
            if (asn1_get_tag_der
              (der + counter, len - counter, &class, &len2,
               &tag) != ASN1_SUCCESS)
            {
              result = ASN1_DER_ERROR;
              goto cleanup;
            }

            if (counter + len2 > len)
            {
              result = ASN1_DER_ERROR;
              goto cleanup;
            }
            len4 =
            asn1_get_length_der (der + counter + len2,
                                 len - counter - len2, &len3);
            if (len4 < -1)
            {
              result = ASN1_DER_ERROR;
              goto cleanup;
            }
            if (len4 != -1)
            {
              len2 += len4;
              _asn1_set_value_lv (p, der + counter, len2 + len3);
              counter += len2 + len3;
            }
            else
            {           /* indefinite length */
              /* Check indefinite lenth method in an EXPLICIT TAG */
              if ((p->type & CONST_TAG) && (der[counter - 1] == 0x80))
                indefinite = 1;
              else
                indefinite = 0;

              len2 = len - counter;
              result =
                _asn1_get_indefinite_length_string (der + counter,
&len2);
              if (result != ASN1_SUCCESS)
                goto cleanup;

              _asn1_set_value_lv (p, der + counter, len2);
              counter += len2;

              /* Check if a couple of 0x00 are present due to an EXPLICIT
TAG with
                 an indefinite length method. */
              if (indefinite)
                {
                  if (!der[counter] && !der[counter + 1])
                  {
                    counter += 2;
                  }
```

```
                else
                {
                  result = ASN1_DER_ERROR;
                  goto cleanup;
                }
              }
            }
            move = RIGHT;
            break;
          default:
            move = (move == UP) ? RIGHT : DOWN;
            break;
          }
      }

      if (p == node && move != DOWN)
      break;

      if (move == DOWN)
      {
        if (p->down)
          p = p->down;
        else
          move = RIGHT;
      }
      if ((move == RIGHT) && !(p->type & CONST_SET))
      {
        if (p->right)
          p = p->right;
        else
          move = UP;
      }
      if (move == UP)
      p = _asn1_find_up (p);
    }

  _asn1_delete_not_used (*element);

  if (counter != len)
    {
      result = ASN1_DER_ERROR;
      goto cleanup;
    }

  return ASN1_SUCCESS;

cleanup:
  asn1_delete_structure (element);
  return result;
}
<sep>
int dccp_recvmsg(struct kiocb *iocb, struct sock *sk, struct msghdr *msg,
            size_t len, int nonblock, int flags, int *addr_len)
{
```

```
	const struct dccp_hdr *dh;
	long timeo;

	lock_sock(sk);

	if (sk->sk_state == DCCP_LISTEN) {
		len = -ENOTCONN;
		goto out;
	}

	timeo = sock_rcvtimeo(sk, nonblock);

	do {
		struct sk_buff *skb = skb_peek(&sk->sk_receive_queue);

		if (skb == NULL)
			goto verify_sock_status;

		dh = dccp_hdr(skb);

		switch (dh->dccph_type) {
		case DCCP_PKT_DATA:
		case DCCP_PKT_DATAACK:
			goto found_ok_skb;

		case DCCP_PKT_CLOSE:
		case DCCP_PKT_CLOSEREQ:
			if (!(flags & MSG_PEEK))
				dccp_finish_passive_close(sk);
			/* fall through */
		case DCCP_PKT_RESET:
			dccp_pr_debug("found fin (%s) ok!\n",
					dccp_packet_name(dh->dccph_type));
			len = 0;
			goto found_fin_ok;
		default:
			dccp_pr_debug("packet_type=%s\n",
					dccp_packet_name(dh->dccph_type));
			sk_eat_skb(sk, skb, false);
		}
verify_sock_status:
		if (sock_flag(sk, SOCK_DONE)) {
			len = 0;
			break;
		}

		if (sk->sk_err) {
			len = sock_error(sk);
			break;
		}

		if (sk->sk_shutdown & RCV_SHUTDOWN) {
			len = 0;
			break;
```

```
            }

            if (sk->sk_state == DCCP_CLOSED) {
                  if (!sock_flag(sk, SOCK_DONE)) {
                        /* This occurs when user tries to read
                         * from never connected socket.
                         */
                        len = -ENOTCONN;
                        break;
                  }
                  len = 0;
                  break;
            }

            if (!timeo) {
                  len = -EAGAIN;
                  break;
            }

            if (signal_pending(current)) {
                  len = sock_intr_errno(timeo);
                  break;
            }

            sk_wait_data(sk, &timeo);
            continue;
      found_ok_skb:
            if (len > skb->len)
                  len = skb->len;
            else if (len < skb->len)
                  msg->msg_flags |= MSG_TRUNC;

            if (skb_copy_datagram_iovec(skb, 0, msg->msg_iov, len)) {
                  /* Exception. Bailout! */
                  len = -EFAULT;
                  break;
            }
            if (flags & MSG_TRUNC)
                  len = skb->len;
      found_fin_ok:
            if (!(flags & MSG_PEEK))
                  sk_eat_skb(sk, skb, false);
            break;
      } while (1);
out:
      release_sock(sk);
      return len;
}
<sep>
sctp_disposition_t sctp_sf_eat_fwd_tsn_fast(
      const struct sctp_endpoint *ep,
      const struct sctp_association *asoc,
      const sctp_subtype_t type,
      void *arg,
```

```
      sctp_cmd_seq_t *commands)
{
      struct sctp_chunk *chunk = arg;
      struct sctp_fwdtsn_hdr *fwdtsn_hdr;
      __u16 len;
      __u32 tsn;

      if (!sctp_vtag_verify(chunk, asoc)) {
            sctp_add_cmd_sf(commands, SCTP_CMD_REPORT_BAD_TAG,
                        SCTP_NULL());
            return sctp_sf_pdiscard(ep, asoc, type, arg, commands);
      }

      /* Make sure that the FORWARD_TSN chunk has a valid length.  */
      if (!sctp_chunk_length_valid(chunk, sizeof(struct
sctp_fwdtsn_chunk)))
            return sctp_sf_violation_chunklen(ep, asoc, type, arg,
                                      commands);

      fwdtsn_hdr = (struct sctp_fwdtsn_hdr *)chunk->skb->data;
      chunk->subh.fwdtsn_hdr = fwdtsn_hdr;
      len = ntohs(chunk->chunk_hdr->length);
      len -= sizeof(struct sctp_chunkhdr);
      skb_pull(chunk->skb, len);

      tsn = ntohl(fwdtsn_hdr->new_cum_tsn);
      SCTP_DEBUG_PRINTK("%s: TSN 0x%x.\n", __func__, tsn);

      /* The TSN is too high--silently discard the chunk and count on it
       * getting retransmitted later.
       */
      if (sctp_tsnmap_check(&asoc->peer.tsn_map, tsn) < 0)
            goto gen_shutdown;

      sctp_add_cmd_sf(commands, SCTP_CMD_REPORT_FWDTSN, SCTP_U32(tsn));
      if (len > sizeof(struct sctp_fwdtsn_hdr))
            sctp_add_cmd_sf(commands, SCTP_CMD_PROCESS_FWDTSN,
                        SCTP_CHUNK(chunk));

      /* Go a head and force a SACK, since we are shutting down. */
gen_shutdown:
      /* Implementor's Guide.
       *
       * While in SHUTDOWN-SENT state, the SHUTDOWN sender MUST
immediately
       * respond to each received packet containing one or more DATA
chunk(s)
       * with a SACK, a SHUTDOWN chunk, and restart the T2-shutdown timer
       */
      sctp_add_cmd_sf(commands, SCTP_CMD_GEN_SHUTDOWN, SCTP_NULL());
      sctp_add_cmd_sf(commands, SCTP_CMD_GEN_SACK, SCTP_FORCE());
      sctp_add_cmd_sf(commands, SCTP_CMD_TIMER_RESTART,
                SCTP_TO(SCTP_EVENT_TIMEOUT_T2_SHUTDOWN));
```

```
     return SCTP_DISPOSITION_CONSUME;
}
<sep>
int
router_differences_are_cosmetic(const routerinfo_t *r1, const
routerinfo_t *r2)
{
  time_t r1pub, r2pub;
  long time_difference;
  tor_assert(r1 && r2);

  /* r1 should be the one that was published first. */
  if (r1->cache_info.published_on > r2->cache_info.published_on) {
    const routerinfo_t *ri_tmp = r2;
    r2 = r1;
    r1 = ri_tmp;
  }

  /* If any key fields differ, they're different. */
  if (r1->addr != r2->addr ||
      strcasecmp(r1->nickname, r2->nickname) ||
      r1->or_port != r2->or_port ||
      !tor_addr_eq(&r1->ipv6_addr, &r2->ipv6_addr) ||
      r1->ipv6_orport != r2->ipv6_orport ||
      r1->dir_port != r2->dir_port ||
      r1->purpose != r2->purpose ||
      !crypto_pk_eq_keys(r1->onion_pkey, r2->onion_pkey) ||
      !crypto_pk_eq_keys(r1->identity_pkey, r2->identity_pkey) ||
      strcasecmp(r1->platform, r2->platform) ||
      (r1->contact_info && !r2->contact_info) || /* contact_info is
optional */
      (!r1->contact_info && r2->contact_info) ||
      (r1->contact_info && r2->contact_info &&
       strcasecmp(r1->contact_info, r2->contact_info)) ||
      r1->is_hibernating != r2->is_hibernating ||
      cmp_addr_policies(r1->exit_policy, r2->exit_policy) ||
      (r1->supports_tunnelled_dir_requests !=
       r2->supports_tunnelled_dir_requests))
    return 0;
  if ((r1->declared_family == NULL) != (r2->declared_family == NULL))
    return 0;
  if (r1->declared_family && r2->declared_family) {
    int i, n;
    if (smartlist_len(r1->declared_family)!=smartlist_len(r2-
>declared_family))
      return 0;
    n = smartlist_len(r1->declared_family);
    for (i=0; i < n; ++i) {
      if (strcasecmp(smartlist_get(r1->declared_family, i),
                     smartlist_get(r2->declared_family, i)))
        return 0;
    }
  }
```

```
  /* Did bandwidth change a lot? */
  if ((r1->bandwidthcapacity < r2->bandwidthcapacity/2) ||
      (r2->bandwidthcapacity < r1->bandwidthcapacity/2))
    return 0;

  /* Did the bandwidthrate or bandwidthburst change? */
  if ((r1->bandwidthrate != r2->bandwidthrate) ||
      (r1->bandwidthburst != r2->bandwidthburst))
    return 0;

  /* Did more than 12 hours pass? */
  if (r1->cache_info.published_on + ROUTER_MAX_COSMETIC_TIME_DIFFERENCE
      < r2->cache_info.published_on)
    return 0;

  /* Did uptime fail to increase by approximately the amount we would
think,
   * give or take some slop? */
  r1pub = r1->cache_info.published_on;
  r2pub = r2->cache_info.published_on;
  time_difference = labs(r2->uptime - (r1->uptime + (r2pub - r1pub)));
  if (time_difference > ROUTER_ALLOW_UPTIME_DRIFT &&
      time_difference > r1->uptime * .05 &&
      time_difference > r2->uptime * .05)
    return 0;

  /* Otherwise, the difference is cosmetic. */
  return 1;
<sep>
next_packet (unsigned char const **bufptr, size_t *buflen,
             unsigned char const **r_data, size_t *r_datalen, int
*r_pkttype,
             size_t *r_ntotal)
{
  const unsigned char *buf = *bufptr;
  size_t len = *buflen;
  int c, ctb, pkttype;
  unsigned long pktlen;

  if (!len)
    return gpg_error (GPG_ERR_NO_DATA);

  ctb = *buf++; len--;
  if ( !(ctb & 0x80) )
    return gpg_error (GPG_ERR_INV_PACKET); /* Invalid CTB. */

  pktlen = 0;
  if ((ctb & 0x40))  /* New style (OpenPGP) CTB.  */
    {
      pkttype = (ctb & 0x3f);
      if (!len)
        return gpg_error (GPG_ERR_INV_PACKET); /* No 1st length byte. */
      c = *buf++; len--;
      if (pkttype == PKT_COMPRESSED)
```

```
        return gpg_error (GPG_ERR_UNEXPECTED); /* ... packet in a
keyblock. */
      if ( c < 192 )
        pktlen = c;
      else if ( c < 224 )
        {
          pktlen = (c - 192) * 256;
          if (!len)
            return gpg_error (GPG_ERR_INV_PACKET); /* No 2nd length byte.
*/
          c = *buf++; len--;
          pktlen += c + 192;
        }
      else if (c == 255)
        {
          if (len <4 )
            return gpg_error (GPG_ERR_INV_PACKET); /* No length bytes. */
          pktlen  = (*buf++) << 24;
          pktlen |= (*buf++) << 16;
          pktlen |= (*buf++) << 8;
          pktlen |= (*buf++);
          len -= 4;
      }
      else /* Partial length encoding is not allowed for key packets. */
        return gpg_error (GPG_ERR_UNEXPECTED);
    }
  else /* Old style CTB.  */
    {
      int lenbytes;

      pktlen = 0;
      pkttype = (ctb>>2)&0xf;
      lenbytes = ((ctb&3)==3)? 0 : (1<<(ctb & 3));
      if (!lenbytes) /* Not allowed in key packets.  */
        return gpg_error (GPG_ERR_UNEXPECTED);
      if (len < lenbytes)
        return gpg_error (GPG_ERR_INV_PACKET); /* Not enough length
bytes.  */
      for (; lenbytes; lenbytes--)
        {
          pktlen <<= 8;
          pktlen |= *buf++; len--;
      }
    }

  /* Do some basic sanity check.  */
  switch (pkttype)
    {
    case PKT_SIGNATURE:
    case PKT_SECRET_KEY:
    case PKT_PUBLIC_KEY:
    case PKT_SECRET_SUBKEY:
    case PKT_MARKER:
    case PKT_RING_TRUST:
```

```
    case PKT_USER_ID:
    case PKT_PUBLIC_SUBKEY:
    case PKT_OLD_COMMENT:
    case PKT_ATTRIBUTE:
    case PKT_COMMENT:
    case PKT_GPG_CONTROL:
      break; /* Okay these are allowed packets. */
    default:
      return gpg_error (GPG_ERR_UNEXPECTED);
    }

  if (pktlen == (unsigned long)(-1))
    return gpg_error (GPG_ERR_INV_PACKET);

  if (pktlen > len)
    return gpg_error (GPG_ERR_INV_PACKET); /* Packet length header too
long. */

  *r_data = buf;
  *r_datalen = pktlen;
  *r_pkttype = pkttype;
  *r_ntotal = (buf - *bufptr) + pktlen;

  *bufptr = buf + pktlen;
  *buflen = len - pktlen;
  if (!*buflen)
    *bufptr = NULL;

  return 0;
}
<sep>
int mbedtls_x509_crt_verify_with_profile( mbedtls_x509_crt *crt,
                     mbedtls_x509_crt *trust_ca,
                     mbedtls_x509_crl *ca_crl,
                     const mbedtls_x509_crt_profile *profile,
                     const char *cn, uint32_t *flags,
                     int (*f_vrfy)(void *, mbedtls_x509_crt *, int,
uint32_t *),
                     void *p_vrfy )
{
    size_t cn_len;
    int ret;
    int pathlen = 0, selfsigned = 0;
    mbedtls_x509_crt *parent;
    mbedtls_x509_name *name;
    mbedtls_x509_sequence *cur = NULL;
    mbedtls_pk_type_t pk_type;

    if( profile == NULL )
        return( MBEDTLS_ERR_X509_BAD_INPUT_DATA );

    *flags = 0;

    if( cn != NULL )
```

```
    {
        name = &crt->subject;
        cn_len = strlen( cn );

        if( crt->ext_types & MBEDTLS_X509_EXT_SUBJECT_ALT_NAME )
        {
            cur = &crt->subject_alt_names;

            while( cur != NULL )
            {
                if( cur->buf.len == cn_len &&
                    x509_memcasecmp( cn, cur->buf.p, cn_len ) == 0 )
                    break;

                if( cur->buf.len > 2 &&
                    memcmp( cur->buf.p, "*.", 2 ) == 0 &&
                    x509_check_wildcard( cn, &cur->buf ) == 0 )
                {
                    break;
                }

                cur = cur->next;
            }

            if( cur == NULL )
                *flags |= MBEDTLS_X509_BADCERT_CN_MISMATCH;
        }
        else
        {
            while( name != NULL )
            {
                if( MBEDTLS_OID_CMP( MBEDTLS_OID_AT_CN, &name->oid ) == 0
)
                {
                    if( name->val.len == cn_len &&
                        x509_memcasecmp( name->val.p, cn, cn_len ) == 0 )
                        break;

                    if( name->val.len > 2 &&
                        memcmp( name->val.p, "*.", 2 ) == 0 &&
                        x509_check_wildcard( cn, &name->val ) == 0 )
                        break;
                }

                name = name->next;
            }

            if( name == NULL )
                *flags |= MBEDTLS_X509_BADCERT_CN_MISMATCH;
        }
    }

    /* Check the type and size of the key */
    pk_type = mbedtls_pk_get_type( &crt->pk );
```

```
    if( x509_profile_check_pk_alg( profile, pk_type ) != 0 )
        *flags |= MBEDTLS_X509_BADCERT_BAD_PK;

    if( x509_profile_check_key( profile, pk_type, &crt->pk ) != 0 )
        *flags |= MBEDTLS_X509_BADCERT_BAD_KEY;

    /* Look for a parent in trusted CAs */
    for( parent = trust_ca; parent != NULL; parent = parent->next )
    {
        if( x509_crt_check_parent( crt, parent, 0, pathlen == 0 ) == 0 )
            break;
    }

    if( parent != NULL )
    {
        ret = x509_crt_verify_top( crt, parent, ca_crl, profile,
                                   pathlen, selfsigned, flags, f_vrfy,
p_vrfy );
        if( ret != 0 )
            return( ret );
    }
    else
    {
        /* Look for a parent upwards the chain */
        for( parent = crt->next; parent != NULL; parent = parent->next )
            if( x509_crt_check_parent( crt, parent, 0, pathlen == 0 ) ==
0 )
                break;

        /* Are we part of the chain or at the top? */
        if( parent != NULL )
        {
            ret = x509_crt_verify_child( crt, parent, trust_ca, ca_crl,
profile,
                                         pathlen, selfsigned, flags,
f_vrfy, p_vrfy );
            if( ret != 0 )
                return( ret );
        }
        else
        {
            ret = x509_crt_verify_top( crt, trust_ca, ca_crl, profile,
                                       pathlen, selfsigned, flags,
f_vrfy, p_vrfy );
            if( ret != 0 )
                return( ret );
        }
    }

    if( *flags != 0 )
        return( MBEDTLS_ERR_X509_CERT_VERIFY_FAILED );

    return( 0 );
```

```
}
<sep>
BGD_DECLARE(gdImagePtr) gdImageCreate (int sx, int sy)
{
      int i;
      gdImagePtr im;

      if (overflow2(sx, sy)) {
            return NULL;
      }
      if (overflow2(sizeof (unsigned char *), sy)) {
            return NULL;
      }
      if (overflow2(sizeof (unsigned char), sx)) {
            return NULL;
      }

      im = (gdImage *) gdCalloc(1, sizeof(gdImage));
      if (!im) {
            return NULL;
      }

      /* Row-major ever since gd 1.3 */
      im->pixels = (unsigned char **) gdMalloc (sizeof (unsigned char *)
* sy);
      if (!im->pixels) {
            gdFree(im);
            return NULL;
      }

      im->polyInts = 0;
      im->polyAllocated = 0;
      im->brush = 0;
      im->tile = 0;
      im->style = 0;
      for (i = 0; (i < sy); i++) {
            /* Row-major ever since gd 1.3 */
            im->pixels[i] = (unsigned char *) gdCalloc (sx, sizeof
(unsigned char));
            if (!im->pixels[i]) {
                  for (--i ; i >= 0; i--) {
                        gdFree(im->pixels[i]);
                  }
                  gdFree(im->pixels);
                  gdFree(im);
                  return NULL;
            }

      }
      im->sx = sx;
      im->sy = sy;
      im->colorsTotal = 0;
      im->transparent = (-1);
      im->interlace = 0;
```

```
      im->thick = 1;
      im->AA = 0;
      for (i = 0; (i < gdMaxColors); i++) {
            im->open[i] = 1;
      };
      im->trueColor = 0;
      im->tpixels = 0;
      im->cx1 = 0;
      im->cy1 = 0;
      im->cx2 = im->sx - 1;
      im->cy2 = im->sy - 1;
      im->res_x = GD_RESOLUTION;
      im->res_y = GD_RESOLUTION;
      im->interpolation = NULL;
      im->interpolation_id = GD_BILINEAR_FIXED;
      return im;
}
<sep>
lyd_new_output_leaf(struct lyd_node *parent, const struct lys_module
*module, const char *name, const char *val_str)
{
    const struct lys_node *snode = NULL, *siblings;

    if ((!parent && !module) || !name) {
        LOGARG;
        return NULL;
    }

    siblings = lyd_new_find_schema(parent, module, 1);
    if (!siblings) {
        LOGARG;
        return NULL;
    }

    if (lys_getnext_data(module, lys_parent(siblings), name,
strlen(name), LYS_LEAFLIST | LYS_LEAF, &snode) || !snode) {
        LOGERR(siblings->module->ctx, LY_EINVAL, "Failed to find \"%s\"
as a sibling to \"%s:%s\".",
               name, lys_node_module(siblings)->name, siblings->name);
        return NULL;
    }

    return _lyd_new_leaf(parent, snode, val_str, 0, 0);
}
<sep>
gst_mpegts_section_new (guint16 pid, guint8 * data, gsize data_size)
{
  GstMpegtsSection *res = NULL;
  guint8 tmp;
  guint8 table_id;
  guint16 section_length;

  /* Check for length */
  section_length = GST_READ_UINT16_BE (data + 1) & 0x0FFF;
```

```
  if (G_UNLIKELY (data_size < section_length + 3))
    goto short_packet;

  /* Table id is in first byte */
  table_id = *data;

  res = _gst_mpegts_section_init (pid, table_id);

  res->data = data;
  /* table_id (already parsed)       : 8  bit */
  data++;
  /* section_syntax_indicator        : 1  bit
   * other_fields (reserved)         : 3  bit*/
  res->short_section = (*data & 0x80) == 0x00;
  /* section_length (already parsed) : 12 bit */
  res->section_length = section_length + 3;
  if (!res->short_section) {
    /* CRC is after section_length (-4 for the size of the CRC) */
    res->crc = GST_READ_UINT32_BE (res->data + res->section_length - 4);
    /* Skip to after section_length */
    data += 2;
    /* subtable extension            : 16 bit */
    res->subtable_extension = GST_READ_UINT16_BE (data);
    data += 2;
    /* reserved                      : 2  bit
     * version_number                : 5  bit
     * current_next_indicator        : 1  bit */
    tmp = *data++;
    res->version_number = tmp >> 1 & 0x1f;
    res->current_next_indicator = tmp & 0x01;
    /* section_number                : 8  bit */
    res->section_number = *data++;
    /* last_section_number                : 8  bit */
    res->last_section_number = *data;
  }

  return res;

short_packet:
  {
    GST_WARNING
        ("PID 0x%04x section extends past provided data (got:%"
G_GSIZE_FORMAT
        ", need:%d)", pid, data_size, section_length + 3);
    g_free (data);
    return NULL;
  }
}
<sep>
arch_get_unmapped_area(struct file *filp, unsigned long addr,
            unsigned long len, unsigned long pgoff, unsigned long flags)
{
      struct mm_struct *mm = current->mm;
      struct vm_area_struct *vma;
```

```
      struct vm_unmapped_area_info info;
      unsigned long begin, end;

      if (flags & MAP_FIXED)
            return addr;

      find_start_end(flags, &begin, &end);

      if (len > end)
            return -ENOMEM;

      if (addr) {
            addr = PAGE_ALIGN(addr);
            vma = find_vma(mm, addr);
            if (end - len >= addr &&
                (!vma || addr + len <= vma->vm_start))
                  return addr;
      }

      info.flags = 0;
      info.length = len;
      info.low_limit = begin;
      info.high_limit = end;
      info.align_mask = 0;
      info.align_offset = pgoff << PAGE_SHIFT;
      if (filp) {
            info.align_mask = get_align_mask();
            info.align_offset += get_align_bits();
      }
      return vm_unmapped_area(&info);
}
<sep>
ofputil_pull_ofp15_group_mod(struct ofpbuf *msg, enum ofp_version
ofp_version,
                             struct ofputil_group_mod *gm)
{
    const struct ofp15_group_mod *ogm;
    uint16_t bucket_list_len;
    enum ofperr error = OFPERR_OFPGMFC_BAD_BUCKET;

    ogm = ofpbuf_pull(msg, sizeof *ogm);
    gm->command = ntohs(ogm->command);
    gm->type = ogm->type;
    gm->group_id = ntohl(ogm->group_id);

    gm->command_bucket_id = ntohl(ogm->command_bucket_id);
    switch (gm->command) {
    case OFPGC15_REMOVE_BUCKET:
        if (gm->command_bucket_id == OFPG15_BUCKET_ALL) {
            error = 0;
        }
        /* Fall through */
    case OFPGC15_INSERT_BUCKET:
        if (gm->command_bucket_id <= OFPG15_BUCKET_MAX ||
```

```
            gm->command_bucket_id == OFPG15_BUCKET_FIRST
            || gm->command_bucket_id == OFPG15_BUCKET_LAST) {
            error = 0;
        }
        break;

    case OFPGC11_ADD:
    case OFPGC11_MODIFY:
    case OFPGC11_ADD_OR_MOD:
    case OFPGC11_DELETE:
    default:
        if (gm->command_bucket_id == OFPG15_BUCKET_ALL) {
            error = 0;
        }
        break;
    }
    if (error) {
        VLOG_WARN_RL(&bad_ofmsg_rl,
                     "group command bucket id (%u) is out of range",
                     gm->command_bucket_id);
        return OFPERR_OFPGMFC_BAD_BUCKET;
    }

    bucket_list_len = ntohs(ogm->bucket_array_len);
    if (bucket_list_len > msg->size) {
        return OFPERR_OFPBRC_BAD_LEN;
    }
    error = ofputil_pull_ofp15_buckets(msg, bucket_list_len, ofp_version,
                                       gm->type, &gm->buckets);
    if (error) {
        return error;
    }

    return parse_ofp15_group_properties(msg, gm->type, gm->command, &gm-
>props,
                                        msg->size);
}
<sep>
mkstemp_helper (struct obstack *obs, const char *name)
{
  int fd;
  int len;
  int i;

  /* Guarantee that there are six trailing 'X' characters, even if the
     user forgot to supply them.  */
  len = strlen (name);
  obstack_grow (obs, name, len);
  for (i = 0; len > 0 && i < 6; i++)
    if (name[--len] != 'X')
      break;
  for (; i < 6; i++)
    obstack_1grow (obs, 'X');
  obstack_1grow (obs, '\0');
```

```
  errno = 0;
  fd = mkstemp ((char *) obstack_base (obs));
  if (fd < 0)
    {
      M4ERROR ((0, errno, "cannot create tempfile `%s'", name));
      obstack_free (obs, obstack_finish (obs));
    }
  else
    close (fd);
}
<sep>
MagickPrivate ResizeFilter *AcquireResizeFilter(const Image *image,
  const FilterType filter,const MagickBooleanType cylindrical,
  ExceptionInfo *exception)
{
  const char
    *artifact;

  FilterType
    filter_type,
    window_type;

  double
    B,
    C,
    value;

  register ResizeFilter
    *resize_filter;

  /*
    Table Mapping given Filter, into Weighting and Windowing functions.
    A 'Box' windowing function means its a simble non-windowed filter.
    An 'SincFast' filter function could be upgraded to a 'Jinc' filter if
a
    "cylindrical" is requested, unless a 'Sinc' or 'SincFast' filter was
    specifically requested by the user.

    WARNING: The order of this table must match the order of the
FilterType
    enumeration specified in "resample.h", or the filter names will not
match
    the filter being setup.

    You can check filter setups with the "filter:verbose" expert setting.
  */
  static struct
  {
    FilterType
      filter,
      window;
  } const mapping[SentinelFilter] =
  {
```

```
    { UndefinedFilter,     BoxFilter      },  /* Undefined (default to
Box)   */
    { PointFilter,         BoxFilter      },  /* SPECIAL: Nearest
neighbour   */
    { BoxFilter,           BoxFilter      },  /* Box averaging filter
*/
    { TriangleFilter,      BoxFilter      },  /* Linear interpolation
filter  */
    { HermiteFilter,       BoxFilter      },  /* Hermite interpolation
filter */
    { SincFastFilter,      HannFilter     },  /* Hann -- cosine-sinc
*/
    { SincFastFilter,      HammingFilter  },  /* Hamming --   ''
variation    */
    { SincFastFilter,      BlackmanFilter },  /* Blackman -- 2*cosine-
sinc    */
    { GaussianFilter,      BoxFilter      },  /* Gaussian blur filter
*/
    { QuadraticFilter,     BoxFilter      },  /* Quadratic Gaussian
approx    */
    { CubicFilter,         BoxFilter      },  /* General Cubic Filter,
Spline */
    { CatromFilter,        BoxFilter      },  /* Cubic-Keys interpolator
*/
    { MitchellFilter,      BoxFilter      },  /* 'Ideal' Cubic-Keys
filter    */
    { JincFilter,          BoxFilter      },  /* Raw 3-lobed Jinc
function    */
    { SincFilter,          BoxFilter      },  /* Raw 4-lobed Sinc
function    */
    { SincFastFilter,      BoxFilter      },  /* Raw fast sinc ("Pade"-
type)  */
    { SincFastFilter,      KaiserFilter   },  /* Kaiser -- square root-
sinc   */
    { LanczosFilter,       WelchFilter    },  /* Welch -- parabolic (3
lobe)  */
    { SincFastFilter,      CubicFilter    },  /* Parzen -- cubic-sinc
*/
    { SincFastFilter,      BohmanFilter   },  /* Bohman -- 2*cosine-sinc
*/
    { SincFastFilter,      TriangleFilter },  /* Bartlett -- triangle-
sinc    */
    { LagrangeFilter,      BoxFilter      },  /* Lagrange self-windowing
*/
    { LanczosFilter,       LanczosFilter  },  /* Lanczos Sinc-Sinc
filters    */
    { LanczosSharpFilter,  LanczosSharpFilter }, /* | these require */
    { Lanczos2Filter,      Lanczos2Filter },     /* | special handling */
    { Lanczos2SharpFilter, Lanczos2SharpFilter },
    { RobidouxFilter,      BoxFilter      },  /* Cubic Keys tuned for EWA
*/
    { RobidouxSharpFilter, BoxFilter      },  /* Sharper Cubic Keys for
EWA   */
```

```
    { LanczosFilter,       CosineFilter   },  /* Cosine window (3 lobes)
*/
    { SplineFilter,        BoxFilter      },  /* Spline Cubic Filter
*/
    { LanczosRadiusFilter, LanczosFilter  },  /* Lanczos with integer
radius  */
    { CubicSplineFilter,   BoxFilter      },  /* CubicSpline (2/3/4
lobes)    */
  };
  /*
    Table mapping the filter/window from the above table to an actual
function.
    The default support size for that filter as a weighting function, the
range
    to scale with to use that function as a sinc windowing function, (typ
1.0).

    Note that the filter_type -> function is 1 to 1 except for Sinc(),
    SincFast(), and CubicBC() functions, which may have multiple filter
to
    function associations.

    See "filter:verbose" handling below for the function -> filter
mapping.
  */
  static struct
  {
    double
      (*function)(const double,const ResizeFilter*),
      support, /* Default lobes/support size of the weighting filter. */
      scale,   /* Support when function used as a windowing function
                  Typically equal to the location of the first zero
crossing. */
      B,C;     /* BC-spline coefficients, ignored if not a CubicBC
filter. */
    ResizeWeightingFunctionType weightingFunctionType;
  } const filters[SentinelFilter] =
  {
    /*            .---  support window (if used as a Weighting Function)
                  |    .--- first crossing (if used as a Windowing
Function)
                  |    |    .--- B value for Cubic Function
                  |    |    |    .---- C value for Cubic Function
                  |    |    |    |                                    */
    { Box,       0.5, 0.5, 0.0, 0.0, BoxWeightingFunction },      /*
Undefined (default to Box)  */
    { Box,       0.0, 0.5, 0.0, 0.0, BoxWeightingFunction },      /*
Point (special handling)    */
    { Box,       0.5, 0.5, 0.0, 0.0, BoxWeightingFunction },      /* Box
*/
    { Triangle,  1.0, 1.0, 0.0, 0.0, TriangleWeightingFunction }, /*
Triangle                    */
    { CubicBC,   1.0, 1.0, 0.0, 0.0, CubicBCWeightingFunction },  /*
Hermite (cubic  B=C=0)      */
```

```
    { Hann,      1.0, 1.0, 0.0, 0.0, HannWeightingFunction },     /*
Hann, cosine window         */
    { Hamming,   1.0, 1.0, 0.0, 0.0, HammingWeightingFunction },  /*
Hamming, '' variation       */
    { Blackman,  1.0, 1.0, 0.0, 0.0, BlackmanWeightingFunction }, /*
Blackman, 2*cosine window   */
    { Gaussian,  2.0, 1.5, 0.0, 0.0, GaussianWeightingFunction }, /*
Gaussian                    */
    { Quadratic, 1.5, 1.5, 0.0, 0.0, QuadraticWeightingFunction },/*
Quadratic gaussian          */
    { CubicBC,   2.0, 2.0, 1.0, 0.0, CubicBCWeightingFunction },  /*
General Cubic Filter        */
    { CubicBC,   2.0, 1.0, 0.0, 0.5, CubicBCWeightingFunction },  /*
Catmull-Rom    (B=0,C=1/2)  */
    { CubicBC,   2.0, 8.0/7.0, 1./3., 1./3., CubicBCWeightingFunction },
/* Mitchell   (B=C=1/3)    */
    { Jinc,      3.0, 1.2196698912665045, 0.0, 0.0, JincWeightingFunction
}, /* Raw 3-lobed Jinc */
    { Sinc,      4.0, 1.0, 0.0, 0.0, SincWeightingFunction },     /* Raw
4-lobed Sinc            */
    { SincFast,  4.0, 1.0, 0.0, 0.0, SincFastWeightingFunction }, /* Raw
fast sinc ("Pade"-type) */
    { Kaiser,    1.0, 1.0, 0.0, 0.0, KaiserWeightingFunction },   /*
Kaiser (square root window) */
    { Welch,     1.0, 1.0, 0.0, 0.0, WelchWeightingFunction },    /*
Welch (parabolic window)    */
    { CubicBC,   2.0, 2.0, 1.0, 0.0, CubicBCWeightingFunction },  /*
Parzen (B-Spline window)    */
    { Bohman,    1.0, 1.0, 0.0, 0.0, BohmanWeightingFunction },   /*
Bohman, 2*Cosine window     */
    { Triangle,  1.0, 1.0, 0.0, 0.0, TriangleWeightingFunction }, /*
Bartlett (triangle window)  */
    { Lagrange,  2.0, 1.0, 0.0, 0.0, LagrangeWeightingFunction }, /*
Lagrange sinc approximation */
    { SincFast,  3.0, 1.0, 0.0, 0.0, SincFastWeightingFunction }, /*
Lanczos, 3-lobed Sinc-Sinc  */
    { SincFast,  3.0, 1.0, 0.0, 0.0, SincFastWeightingFunction }, /*
Lanczos, Sharpened          */
    { SincFast,  2.0, 1.0, 0.0, 0.0, SincFastWeightingFunction }, /*
Lanczos, 2-lobed            */
    { SincFast,  2.0, 1.0, 0.0, 0.0, SincFastWeightingFunction }, /*
Lanczos2, sharpened         */
    /* Robidoux: Keys cubic close to Lanczos2D sharpened */
    { CubicBC,   2.0, 1.1685777620836932,
                           0.37821575509399867, 0.31089212245300067,
CubicBCWeightingFunction },
    /* RobidouxSharp: Sharper version of Robidoux */
    { CubicBC,   2.0, 1.105822933719019,
                           0.2620145123990142,  0.3689927438004929,
CubicBCWeightingFunction },
    { Cosine,    1.0, 1.0, 0.0, 0.0, CosineWeightingFunction },   /* Low
level cosine window     */
    { CubicBC,   2.0, 2.0, 1.0, 0.0, CubicBCWeightingFunction },  /*
Cubic B-Spline (B=1,C=0)    */
```

```
    { SincFast,  3.0, 1.0, 0.0, 0.0, SincFastWeightingFunction }, /*
Lanczos, Interger Radius    */
    { CubicSpline,2.0, 0.5, 0.0, 0.0, BoxWeightingFunction },  /* Spline
Lobes 2-lobed */
  };
  /*
    The known zero crossings of the Jinc() or more accurately the
Jinc(x*PI)
    function being used as a filter. It is used by the "filter:lobes"
expert
    setting and for 'lobes' for Jinc functions in the previous table.
This way
    users do not have to deal with the highly irrational lobe sizes of
the Jinc
    filter.

    Values taken from
    http://cose.math.bas.bg/webMathematica/webComputing/BesselZeros.jsp
    using Jv-function with v=1, then dividing by PI.
  */
  static double
    jinc_zeros[16] =
    {
      1.2196698912665045,
      2.2331305943815286,
      3.2383154841662362,
      4.2410628637960699,
      5.2427643768701817,
      6.2439216898644877,
      7.2447598687199570,
      8.2453949139520427,
      9.2458926849494673,
      10.246293348754916,
      11.246622794877883,
      12.246898461138105,
      13.247132522181061,
      14.247333735806849,
      15.247508563037300,
      16.247661874700962
   };

  /*
    Allocate resize filter.
  */
  assert(image != (const Image *) NULL);
  assert(image->signature == MagickCoreSignature);
  if (image->debug != MagickFalse)
    (void) LogMagickEvent(TraceEvent,GetMagickModule(),"%s",image-
>filename);
  assert(UndefinedFilter < filter && filter < SentinelFilter);
  assert(exception != (ExceptionInfo *) NULL);
  assert(exception->signature == MagickCoreSignature);
  (void) exception;
```

```
  resize_filter=(ResizeFilter *)
AcquireCriticalMemory(sizeof(*resize_filter));
  (void) memset(resize_filter,0,sizeof(*resize_filter));
  /*
    Defaults for the requested filter.
  */
  filter_type=mapping[filter].filter;
  window_type=mapping[filter].window;
  resize_filter->blur=1.0;
  /* Promote 1D Windowed Sinc Filters to a 2D Windowed Jinc filters */
  if ((cylindrical != MagickFalse) && (filter_type == SincFastFilter) &&
      (filter != SincFastFilter))
    filter_type=JincFilter;  /* 1D Windowed Sinc => 2D Windowed Jinc
filters */

  /* Expert filter setting override */
  artifact=GetImageArtifact(image,"filter:filter");
  if (IsStringTrue(artifact) != MagickFalse)
    {
      ssize_t
        option;


option=ParseCommandOption(MagickFilterOptions,MagickFalse,artifact);
      if ((UndefinedFilter < option) && (option < SentinelFilter))
        { /* Raw filter request - no window function. */
          filter_type=(FilterType) option;
          window_type=BoxFilter;
        }
      /* Filter override with a specific window function. */
      artifact=GetImageArtifact(image,"filter:window");
      if (artifact != (const char *) NULL)
        {

option=ParseCommandOption(MagickFilterOptions,MagickFalse,artifact);
          if ((UndefinedFilter < option) && (option < SentinelFilter))
            window_type=(FilterType) option;
        }
    }
  else
    {
      /* Window specified, but no filter function?  Assume Sinc/Jinc. */
      artifact=GetImageArtifact(image,"filter:window");
      if (artifact != (const char *) NULL)
        {
          ssize_t
            option;


option=ParseCommandOption(MagickFilterOptions,MagickFalse,artifact);
          if ((UndefinedFilter < option) && (option < SentinelFilter))
            {
              filter_type= cylindrical != MagickFalse ? JincFilter
                                                      : SincFastFilter;
```

```
              window_type=(FilterType) option;
            }
        }
    }

  /* Assign the real functions to use for the filters selected. */
  resize_filter->filter=filters[filter_type].function;
  resize_filter->support=filters[filter_type].support;
  resize_filter-
>filterWeightingType=filters[filter_type].weightingFunctionType;
  resize_filter->window=filters[window_type].function;
  resize_filter-
>windowWeightingType=filters[window_type].weightingFunctionType;
  resize_filter->scale=filters[window_type].scale;
  resize_filter->signature=MagickCoreSignature;

  /* Filter Modifications for orthogonal/cylindrical usage */
  if (cylindrical != MagickFalse)
    switch (filter_type)
    {
      case BoxFilter:
        /* Support for Cylindrical Box should be sqrt(2)/2 */
        resize_filter->support=(double) MagickSQ1_2;
        break;
      case LanczosFilter:
      case LanczosSharpFilter:
      case Lanczos2Filter:
      case Lanczos2SharpFilter:
      case LanczosRadiusFilter:
        resize_filter->filter=filters[JincFilter].function;
        resize_filter->window=filters[JincFilter].function;
        resize_filter->scale=filters[JincFilter].scale;
        /* number of lobes (support window size) remain unchanged */
        break;
      default:
        break;
    }
  /* Global Sharpening (regardless of orthoginal/cylindrical) */
  switch (filter_type)
  {
    case LanczosSharpFilter:
      resize_filter->blur *= 0.9812505644269356;
      break;
    case Lanczos2SharpFilter:
      resize_filter->blur *= 0.9549963639785485;
      break;
    /* case LanczosRadius:  blur adjust is done after lobes */
    default:
      break;
  }

  /*
    Expert Option Modifications.
  */
```

```
  /* User Gaussian Sigma Override - no support change */
  if ((resize_filter->filter == Gaussian) ||
      (resize_filter->window == Gaussian) ) {
    value=0.5;    /* guassian sigma default, half pixel */
    artifact=GetImageArtifact(image,"filter:sigma");
    if (artifact != (const char *) NULL)
      value=StringToDouble(artifact,(char **) NULL);
    /* Define coefficents for Gaussian */
    resize_filter->coefficient[0]=value;                /* note sigma
too */
    resize_filter->coefficient[1]=PerceptibleReciprocal(2.0*value*value);
/* sigma scaling */
    resize_filter-
>coefficient[2]=PerceptibleReciprocal(Magick2PI*value*value);
       /* normalization - not actually needed or used! */
    if ( value > 0.5 )
      resize_filter->support *= 2*value;  /* increase support linearly */
  }

  /* User Kaiser Alpha Override - no support change */
  if ((resize_filter->filter == Kaiser) ||
      (resize_filter->window == Kaiser) ) {
    value=6.5; /* default beta value for Kaiser bessel windowing function
*/
    artifact=GetImageArtifact(image,"filter:alpha");  /* FUTURE:
depreciate */
    if (artifact != (const char *) NULL)
      value=StringToDouble(artifact,(char **) NULL);
    artifact=GetImageArtifact(image,"filter:kaiser-beta");
    if (artifact != (const char *) NULL)
      value=StringToDouble(artifact,(char **) NULL);
    artifact=GetImageArtifact(image,"filter:kaiser-alpha");
    if (artifact != (const char *) NULL)
      value=StringToDouble(artifact,(char **) NULL)*MagickPI;
    /* Define coefficents for Kaiser Windowing Function */
    resize_filter->coefficient[0]=value;         /* alpha */
    resize_filter->coefficient[1]=PerceptibleReciprocal(I0(value));
      /* normalization */
  }

  /* Support Overrides */
  artifact=GetImageArtifact(image,"filter:lobes");
  if (artifact != (const char *) NULL)
    {
      ssize_t
        lobes;

      lobes=(ssize_t) StringToLong(artifact);
      if (lobes < 1)
        lobes=1;
      resize_filter->support=(double) lobes;
    }
  if (resize_filter->filter == Jinc)
```

```
    {
      /*
        Convert a Jinc function lobes value to a real support value.
      */
      if (resize_filter->support > 16)
        resize_filter->support=jinc_zeros[15];  /* largest entry in table
*/
      else
        resize_filter->support=jinc_zeros[((long) resize_filter-
>support)-1];
      /*
        Blur this filter so support is a integer value (lobes dependant).
      */
      if (filter_type == LanczosRadiusFilter)
        resize_filter->blur*=floor(resize_filter->support)/
          resize_filter->support;
    }
  /*
    Expert blur override.
  */
  artifact=GetImageArtifact(image,"filter:blur");
  if (artifact != (const char *) NULL)
    resize_filter->blur*=StringToDouble(artifact,(char **) NULL);
  if (resize_filter->blur < MagickEpsilon)
    resize_filter->blur=(double) MagickEpsilon;
  /*
    Expert override of the support setting.
  */
  artifact=GetImageArtifact(image,"filter:support");
  if (artifact != (const char *) NULL)
    resize_filter->support=fabs(StringToDouble(artifact,(char **) NULL));
  /*
    Scale windowing function separately to the support 'clipping' window
    that calling operator is planning to actually use. (Expert override)
  */
  resize_filter->window_support=resize_filter->support; /* default */
  artifact=GetImageArtifact(image,"filter:win-support");
  if (artifact != (const char *) NULL)
    resize_filter->window_support=fabs(StringToDouble(artifact,(char **)
NULL));
  /*
    Adjust window function scaling to match windowing support for
weighting
    function.  This avoids a division on every filter call.
  */
  resize_filter->scale/=resize_filter->window_support;
  /*
   * Set Cubic Spline B,C values, calculate Cubic coefficients.
  */
  B=0.0;
  C=0.0;
  if ((resize_filter->filter == CubicBC) ||
      (resize_filter->window == CubicBC) )
    {
```

```
      B=filters[filter_type].B;
      C=filters[filter_type].C;
      if (filters[window_type].function == CubicBC)
        {
          B=filters[window_type].B;
          C=filters[window_type].C;
        }
      artifact=GetImageArtifact(image,"filter:b");
      if (artifact != (const char *) NULL)
        {
          B=StringToDouble(artifact,(char **) NULL);
          C=(1.0-B)/2.0; /* Calculate C to get a Keys cubic filter. */
          artifact=GetImageArtifact(image,"filter:c"); /* user C override
*/
          if (artifact != (const char *) NULL)
            C=StringToDouble(artifact,(char **) NULL);
        }
      else
        {
          artifact=GetImageArtifact(image,"filter:c");
          if (artifact != (const char *) NULL)
            {
              C=StringToDouble(artifact,(char **) NULL);
              B=1.0-2.0*C; /* Calculate B to get a Keys cubic filter. */
            }
        }
      {
        const double
          twoB = B+B;

        /*
          Convert B,C values into Cubic Coefficents. See CubicBC().
        */
        resize_filter->coefficient[0]=1.0-(1.0/3.0)*B;
        resize_filter->coefficient[1]=-3.0+twoB+C;
        resize_filter->coefficient[2]=2.0-1.5*B-C;
        resize_filter->coefficient[3]=(4.0/3.0)*B+4.0*C;
        resize_filter->coefficient[4]=-8.0*C-twoB;
        resize_filter->coefficient[5]=B+5.0*C;
        resize_filter->coefficient[6]=(-1.0/6.0)*B-C;
      }
    }

  /*
    Expert Option Request for verbose details of the resulting filter.
  */
#if defined(MAGICKCORE_OPENMP_SUPPORT)
  #pragma omp master
  {
#endif
    if (IsStringTrue(GetImageArtifact(image,"filter:verbose")) !=
MagickFalse)
      {
        double
```

```
          support,
          x;

        /*
          Set the weighting function properly when the weighting function
          may not exactly match the filter of the same name.  EG: a Point
          filter is really uses a Box weighting function with a different
          support than is typically used.
        */
        if (resize_filter->filter == Box)       filter_type=BoxFilter;
        if (resize_filter->filter == Sinc)      filter_type=SincFilter;
        if (resize_filter->filter == SincFast)
filter_type=SincFastFilter;
        if (resize_filter->filter == Jinc)      filter_type=JincFilter;
        if (resize_filter->filter == CubicBC)   filter_type=CubicFilter;
        if (resize_filter->window == Box)       window_type=BoxFilter;
        if (resize_filter->window == Sinc)      window_type=SincFilter;
        if (resize_filter->window == SincFast)
window_type=SincFastFilter;
        if (resize_filter->window == Jinc)      window_type=JincFilter;
        if (resize_filter->window == CubicBC)   window_type=CubicFilter;
        /*
          Report Filter Details.
        */
        support=GetResizeFilterSupport(resize_filter);  /*
practical_support */
        (void) FormatLocaleFile(stdout,
          "# Resampling Filter (for graphing)\n#\n");
        (void) FormatLocaleFile(stdout,"# filter = %s\n",
          CommandOptionToMnemonic(MagickFilterOptions,filter_type));
        (void) FormatLocaleFile(stdout,"# window = %s\n",
          CommandOptionToMnemonic(MagickFilterOptions,window_type));
        (void) FormatLocaleFile(stdout,"# support = %.*g\n",
          GetMagickPrecision(),(double) resize_filter->support);
        (void) FormatLocaleFile(stdout,"# window-support = %.*g\n",
          GetMagickPrecision(),(double) resize_filter->window_support);
        (void) FormatLocaleFile(stdout,"# scale-blur = %.*g\n",
          GetMagickPrecision(),(double) resize_filter->blur);
        if ((filter_type == GaussianFilter) || (window_type ==
GaussianFilter))
          (void) FormatLocaleFile(stdout,"# gaussian-sigma = %.*g\n",
            GetMagickPrecision(),(double) resize_filter->coefficient[0]);
        if ( filter_type == KaiserFilter || window_type == KaiserFilter )
          (void) FormatLocaleFile(stdout,"# kaiser-beta = %.*g\n",
            GetMagickPrecision(),(double) resize_filter->coefficient[0]);
        (void) FormatLocaleFile(stdout,"# practical-support = %.*g\n",
          GetMagickPrecision(), (double) support);
        if ((filter_type == CubicFilter) || (window_type == CubicFilter))
          (void) FormatLocaleFile(stdout,"# B,C = %.*g,%.*g\n",
            GetMagickPrecision(),(double) B,GetMagickPrecision(),(double)
C);
        (void) FormatLocaleFile(stdout,"\n");
        /*
```

```
          Output values of resulting filter graph -- for graphing filter
result.
        */
        for (x=0.0; x <= support; x+=0.01f)
          (void) FormatLocaleFile(stdout,"%5.2lf\t%.*g\n",x,
            GetMagickPrecision(),(double)
            GetResizeFilterWeight(resize_filter,x));
        /*
          A final value so gnuplot can graph the 'stop' properly.
        */
        (void) FormatLocaleFile(stdout,"%5.2lf\t%.*g\n",support,
          GetMagickPrecision(),0.0);
      }
      /* Output the above once only for each image - remove setting */
    (void) DeleteImageArtifact((Image *) image,"filter:verbose");
#if defined(MAGICKCORE_OPENMP_SUPPORT)
  }
#endif
  return(resize_filter);
}
<sep>
```