

Руководство программиста в Photon

(Photon Programmer's Guide)

Перевод выполнен Владимиром Зайцевым

Содержание

НЕБОЛЬШОЕ ПОЯСНЕНИЕ ПЕРЕВОДЧИКА	10
ВВЕДЕНИЕ	11
ОБЗОР АРХИТЕКТУРЫ RNOTON'A	11
ПОСТРОИТЕЛЬ RNOTON'ОВСКИХ ПРИЛОЖЕНИЙ – RНАВ	12
КОНЦЕПЦИИ ВИДЖЕТОВ	14
ПАРАДИГМА (СИСТЕМА ПОНЯТИЙ) ПРОГРАММИРОВАНИЯ	20
БИБЛИОТЕКИ RNOTON'A	22
ОБЗОР ПОСТРОЕНИЯ ПРИЛОЖЕНИЙ ПОД RНАВ	24
НАПИСАНИЕ ПРИЛОЖЕНИЙ БЕЗ RНАВ	26
ГЛАВА 1. УРОКИ	27
ПЕРЕД ТЕМ КАК ВАМ НАЧАТЬ	27
УРОК 1 – ПРИВЕТ, МИР	28
<i>Создание приложения</i>	28
<i>Генерирование кода</i>	29
<i>Хотите больше информации?</i>	30
УРОК 2. РЕДАКТИРОВАНИЕ РЕСУРСОВ	30
<i>Добавление виджета "Кнопка"</i>	31
<i>Редактирование пиксельной карты</i>	33
<i>Редактирование многострочного текста</i>	34
<i>Редактирование списка текстовых параграфов</i>	35
<i>Создание шаблона</i>	36
<i>Желаете узнать побольше?</i>	38
УРОК 3. СОЗДАНИЕ МЕНЮ И ПАНЕЛЕЙ МЕНЮ	38
<i>О присоединении ответных реакций</i>	38
<i>Об именах экземпляров</i>	39
<i>Создание панели меню</i>	39
<i>Создание модуля меню "File"</i>	40
<i>Создание модуля меню "Help"</i>	41
<i>Присоединение ответных реакций</i>	41
<i>Подготовка кода</i>	43
<i>Хотите узнать больше?</i>	43
УРОК 4. СОЗДАНИЕ ДИАЛОГОВ	44
<i>О диалогах</i>	44
<i>Ещё об именах экземпляров</i>	44
<i>Прикрепление модуля диалога</i>	44
<i>Добавление виджетов в диалог</i>	45
<i>Добавление ответной реакции к кнопке Done</i>	46
<i>Модификация сгенерированного кода функции</i>	47
<i>Компиляция и запуск на выполнение</i>	47
<i>Желаете узнать больше?</i>	48
УРОК 5. СОЗДАНИЕ ОКОН	48
<i>Создание окна</i>	48
<i>Прикрепление ответной реакции</i>	49
<i>Добавление виджетов</i>	49
<i>Генерирование и модификация кода</i>	50
<i>Компиляция и запуск</i>	53
<i>Хотите узнать больше?</i>	53
ГЛАВА 2. ОКРУЖЕНИЕ RНАВ	54
МЕНЮ	54
ПАНЕЛИ ИНСТРУМЕНТОВ	56
ПАНЕЛИ УПРАВЛЕНИЯ	58
ПАЛИТРА ВИДЖЕТОВ	58

:	3
<hr/>	
ПАНЕЛЬ РЕСУРСОВ	60
ПАНЕЛЬ ОТВЕТНЫХ РЕАКЦИЙ.....	61
ПАНЕЛЬ ДЕРЕВА МОДУЛЕЙ.....	61
ПАНЕЛЬ СВЯЗЕЙ МОДУЛЯ.....	62
ПАНЕЛЬ ПОИСКА.....	63
ПОДГОНКА ВАШЕГО ОКРУЖЕНИЯ RnAB	63
<i>Общие предпочтения ("General preferences")</i>	64
<i>Предпочтение цвета ("Color preferences")</i>	64
<i>Предпочтение по перетаскиванию (Dragging preferences)</i>	65
<i>Предпочтения сетки</i>	65
ГЛАВА 3. РАБОТА С ПРИЛОЖЕНИЯМИ.....	66
Создание приложения	66
Открытие приложения	67
Сохранение приложения	68
<i>Именованное и переименованное приложение</i>	68
<i>Сохранение существующего приложения</i>	69
<i>Переписывание существующего приложения</i>	69
Закрывание приложения.....	69
Задание стартовой информации приложения.....	69
<i>Задание глобального заголовочного файла</i>	70
<i>Функция инициализации</i>	70
<i>Опции командной строки</i>	71
<i>Включение имён экземпляров</i>	71
<i>Окна запуска</i>	72
Импортирование файлов	72
<i>Импортирование модулей RnAB из других приложений</i>	73
<i>Импортирование графических образов</i>	73
ГЛАВА 4. РАБОТА С МОДУЛЯМИ.....	74
Типы модулей	74
Анатомия модуля	75
Выбор модуля	76
Как сохраняются модули	76
Изменение ресурсов модуля.....	76
Использование селектора модулей	76
Создание нового модуля	77
Просмотр модулей	77
Открытие модулей	78
Удаление модуля	78
Сворачивание модулей в иконки	78
Закрывание модуля.....	78
Отображение модулей в реальном времени	79
Нахождение потерянных модулей и иконок.....	80
Модули окон.....	80
Модули диалога.....	81
Модули меню	81
Модули картинок	87
Модули иконок	88
ГЛАВА 5. СОЗДАНИЕ ВИДЖЕТОВ В RnAB.....	89
Типы виджетов	89
Имена экземпляров	90
Создание виджета.....	91
Выбор виджетов	92
Выравнивание виджетов	94
Общепользовательский доступ (СИА) и управление фокусом.....	95
Упорядочивание виджетов.....	97
Перетаскивание виджетов	98
Установка координат x и y виджетов.....	99
Перемещение виджетов между контейнерами	99
Изменение размеров виджетов и модулей.....	99

:	100
БУФЕР ОБМЕНА	100
ДУБЛИРОВАНИЕ ВИДЖЕТОВ И КОНТЕЙНЕРОВ	101
УДАЛЕНИЕ ВИДЖЕТОВ	101
ИМПОРТИРОВАНИЕ ГРАФИЧЕСКИХ ФАЙЛОВ	102
ИЗМЕНЕНИЕ КЛАССА ВИДЖЕТА	102
ШАБЛОНЫ	102
ГЛАВА 6. РЕДАКТИРОВАНИЕ РЕСУРСОВ И ОТВЕТНЫХ РЕАКЦИЙ В РНАВ	105
РЕДАКТИРОВАНИЕ РЕСУРСОВ ВИДЖЕТА	105
ПОПИКСЕЛЬНЫЙ РЕДАКТОР	106
РЕДАКТОР ЦВЕТА	109
РЕДАКТОР ФЛАГОВ/ОПЦИЙ	110
РЕДАКТОР ШРИФТОВ	111
РЕДАКТОР СПИСКА	112
РЕДАКТОР ЧИСЕЛ	113
ТЕКСТОВЫЕ РЕДАКТОРЫ	113
РЕДАКТОР ФУНКЦИЙ	115
ОТВЕТНЫЕ РЕАКЦИИ	115
МОДУЛЬНЫЕ ОТВЕТНЫЕ РЕАКЦИИ	118
ОТВЕТНЫЕ РЕАКЦИИ ГОРЯЧИХ КЛАВИШ	120
<i>Горячие клавиши – основы</i>	120
<i>Задание метки горячей клавиши</i>	121
<i>Задание ответной реакции</i>	121
<i>Обработка горячих клавиш</i>	122
<i>Отключение горячих клавиш</i>	123
ОБРАБОТЧИКИ СОБЫТИЙ – НЕОБРАБОТАННЫЕ И ОТФИЛЬТРОВАННЫЕ ОТВЕТНЫЕ РЕАКЦИИ	123
ГЛАВА 7. УПРАВЛЕНИЕ ГЕОМЕТРИЕЙ	125
КОНТЕЙНЕР ВИДЖЕТОВ	125
СОГЛАСОВАНИЕ ГЕОМЕТРИИ	125
АБСОЛЮТНОЕ ПОЗИЦИОНИРОВАНИЕ	129
ВЫРАВНИВАНИЕ ВИДЖЕТОВ С ИСПОЛЬЗОВАНИЕМ ГРУПП	129
УПРАВЛЕНИЕ ПРИВЯЗКОЙ С ИСПОЛЬЗОВАНИЕМ АНКЕРОВ – СРЕДСТВ ПРИВЯЗКИ	132
УСТАНОВКА ОГРАНИЧЕНИЙ ПО РАЗМЕРАМ ИЛИ ПОЗИЦИОНИРОВАНИЮ БЕЗ АНКЕРОВ	136
ГЛАВА 8. ГЕНЕРИРОВАНИЕ, КОМПИЛИРОВАНИЕ И ЗАПУСК ПРОГРАММНОГО КОДА НА ИСПОЛНЕНИЕ	137
ИСПОЛЬЗОВАНИЕ ДИАЛОГА BUILD+RUN	137
ГЕНЕРИРОВАНИЕ ПРОГРАММНОГО КОДА ПРИЛОЖЕНИЯ	138
КАК ОРГАНИЗОВАТЬ ФАЙЛЫ ПРИЛОЖЕНИЯ	142
РЕДАКТИРОВАНИЕ ИСХОДНОГО КОДА	143
КОМПИЛИРОВАНИЕ И ЛИНКОВКА	144
ЗАПУСК ПРИЛОЖЕНИЯ НА ИСПОЛНЕНИЕ	145
ОТЛАДКА	146
ВКЛЮЧЕНИЕ В ВАШЕ ПРИЛОЖЕНИЕ НЕ-РНАВ ФАЙЛОВ	147
СОЗДАНИЕ РЕЗУЛЬТИРУЮЩЕЙ DLL КАК ПРИЛОЖЕНИЯ РНАВ	147
ГЛАВА 9. РАБОТА С ПРОГРАММНЫМ КОДОМ	150
ПЕРЕМЕННЫЕ И ДЕКЛАРАЦИИ	150
ГЛОБАЛЬНЫЙ ХЕАДЕР-ФАЙЛ	152
ИМЕНА ФУНКЦИЙ И ИМЕНА ФАЙЛОВ	153
ФУНКЦИЯ ИНИЦИАЛИЗАЦИИ	154
УСТАНОВОЧНЫЕ ФУНКЦИИ МОДУЛЯ	156
ФУНКЦИИ ОТВЕТНЫХ РЕАКЦИЙ КОДОВОГО ТИПА	157
ТИПЫ ДАННЫХ ГЕОМЕТРИИ	158
ТАЙМЕРЫ	158
МЕНЮ ИНИЦИАЛИЗАЦИИ	159
ЗАДЕРЖКА И ПРИНУДИТЕЛЬНОЕ ОБНОВЛЕНИЕ ИЗОБРАЖЕНИЯ НА ЭКРАНЕ	164
ГЛАВА 10. МАНИПУЛИРОВАНИЕ РЕСУРСАМИ В КОДЕ ПРИЛОЖЕНИЯ	166
СПИСОК АРГУМЕНТОВ	166
УСТАНОВКА РЕСУРСОВ	167

Получение ресурсов	171
ГЛАВА 11. УПРАВЛЕНИЕ ВИДЖЕТАМИ В ИСХОДНОМ КОДЕ ПРИЛОЖЕНИЯ.....	177
Создание виджетов.....	177
Задание порядка виджетов.....	178
Ответные реакции	179
Обработчики событий.....	182
Стили виджетов	184
ГЛАВА 12. ПОВЕРХНОСТИ УПРАВЛЕНИЯ.....	187
Что такое поверхности управления?	187
Ограничения	187
Привязка действий к поверхностям управления	188
Ссылка на поверхности управления.....	188
API поверхностей управления	189
<i>Создания и уничтожение поверхностей управления.....</i>	<i>189</i>
<i>Нахождения идентификаторов для поверхностей управления.....</i>	<i>189</i>
<i>Вычисление геометрии для поверхностей управления</i>	<i>189</i>
<i>Прорисовка поверхностей управления.....</i>	<i>190</i>
<i>Активация поверхностей управления.....</i>	<i>190</i>
<i>Включение и отключение поверхностей управления.....</i>	<i>190</i>
<i>Нахождение поверхностей управления.....</i>	<i>190</i>
<i>Скрытие и демонстрация поверхностей управления.....</i>	<i>190</i>
<i>Установка порядка поверхностей управления</i>	<i>191</i>
<i>Размещение пользовательских данных вместе с поверхностями управления</i>	<i>191</i>
<i>Пример.....</i>	<i>191</i>
ГЛАВА 13. ДОСТУП К МОДУЛЯМ RNAV ИЗ ПРОГРАММНОГО КОДА.....	193
Создание внутренних связей.....	193
Использование внутренних связей в Вашем программном коде.....	194
Использование базы данных виджетов	196
<i>Создание базы данных.....</i>	<i>196</i>
<i>Создание динамической базы данных</i>	<i>197</i>
<i>Функции базы данных виджетов.....</i>	<i>197</i>
ГЛАВА 14. ПОДДЕРЖКА МЕЖДУНАРОДНЫХ ЯЗЫКОВ.....	200
Соображения о проектировании приложения	200
<i>Размер виджетов, основанных на тексте</i>	<i>200</i>
<i>Выравнивание</i>	<i>201</i>
<i>Высота шрифта.....</i>	<i>202</i>
<i>Жёстко закодированные строки</i>	<i>202</i>
<i>Использование символа "@" в именах экземпляров</i>	<i>203</i>
<i>Двуязычные приложения</i>	<i>203</i>
<i>Общие строки.....</i>	<i>204</i>
Генерация языковой базы данных	204
Базы данных сообщений	204
Редактор языка	205
Запуск Вашего приложения на исполнение	208
Распространение Вашего приложения	209
ГЛАВА 15. КОНТЕКСТНО-ЧУВСТВИТЕЛЬНАЯ ПОМОЩЬ.....	210
Создание текста помощи	210
<i>ТЭГИ.....</i>	<i>210</i>
<i>АТТРИБУТЫ.....</i>	<i>210</i>
Ссылки на темы помощи	212
Связывание помощи с виджетами	213
Получение доступа к помощи из Вашего программного кода	214
ГЛАВА 16. МЕЖПРОЦЕССНЫЕ СВЯЗИ	215
Коннекции	216
<i>Пример.....</i>	<i>218</i>
Отсылка QNX-сообщений.....	219

ПРИЁМ QNX-СООБЩЕНИЙ	220
<i>Пример – регистрация сообщений об ошибках</i>	223
ИМПУЛЬСЫ PHOTON'A	224
<i>Пример – очередь сообщений</i>	226
ОБРАБОТКА СИГНАЛОВ	228
ДРУГИЕ МЕХАНИЗМЫ ВВОДА/ВЫВОДА	229
ГЛАВА 17. ПАРАЛЛЕЛЬНЫЕ ОПЕРАЦИИ	230
ОБЗОР	230
ФОНОВОЕ ИСПОЛНЕНИЕ	230
РАБОЧИЕ ПРОЦЕДУРЫ	231
ПОТОКИ	234
<i>Запирание библиотеки Photon'a</i>	235
<i>Несколько потоков, обрабатывающих события</i>	236
<i>Потоки реального времени</i>	236
<i>Не-Photon'овские и Photon'овские потоки</i>	237
<i>Модальные операции и потоки</i>	237
<i>Завершение многопоточной программы</i>	238
<i>Потоки и рабочие процедуры</i>	239
ГЛАВА 18. НЕОБРАБОТАННОЕ РИСОВАНИЕ И МУЛЬТИПЛИКАЦИЯ	240
ВИДЖЕТ PtRAW	240
<i>Функция необработанного рисования</i>	241
<i>Определение холста необработанного виджета</i>	242
<i>Преобразование координат</i>	242
<i>Отсечение</i>	242
<i>Использование повреждённых черепиц (tiles)</i>	243
<i>Использование модели для более сложного рисования</i>	244
<i>Примеры простых функций прорисовки PtRaw</i>	244
ЦВЕТ	245
АТТРИБУТЫ РИСОВАНИЯ	246
ДУГИ, ЭЛЛИПСЫ, МНОГОУГОЛЬНИКИ И ПРЯМОУГОЛЬНИКИ	247
<i>Прямоугольники</i>	248
<i>Многоугольники</i>	250
<i>Дуги, круги, хорды и сектора</i>	251
СПЭНЫ – СЛОЖНЫЕ КРИВЫЕ	252
<i>Линии, пиксели и массивы пикселей</i>	252
ТЕКСТ	253
ПОБИТОВЫЕ ОБРАЗЫ (BITMAPS)	254
ОБРАЗЫ (IMAGES)	255
<i>Образы на основе палитры</i>	256
<i>Образы в непосредственных цветах</i>	256
<i>Образы в градиентных цветах</i>	256
<i>Создание образов</i>	256
<i>Кэширование образов</i>	257
<i>Прозрачность в образах</i>	257
<i>Отображение образов</i>	258
<i>Управление образами</i>	258
<i>Освобождение образов</i>	258
МУЛЬТИПЛИКАЦИЯ	259
<i>Создание серии кадров</i>	259
<i>Циклическая прокрутка кадров</i>	260
<i>Исключение мерцания в мультипликации</i>	261
РЕЖИМ РИСОВАНИЯ С ПРЯМЫМ ДОСТУПОМ	262
ВНЕЭКРАННАЯ ВИДЕОПАМЯТЬ	264
ПОДДЕРЖКА АЛЬФА-СОПРЯЖЕНИЯ	266
ПОДДЕРЖКА ХРОМАТИЧЕСКОГО КЛЮЧА	267
ОПЕРАЦИИ РАСШИРЕННОГО РАСТРА	267
ВИДЕОРЕЖИМЫ	268
ГРАДИЕНТЫ	269
ВИДЕОВЕРЛЕЙ	269
СЛОИ	272
<i>Поверхности</i>	272

:	7
<i>Окна просмотра</i>	273
<i>API слоев</i>	273
<i>Пример</i>	274
ГЛАВА 19. ШРИФТЫ	277
МЕТРИКИ СИМВОЛА	277
ИМЕНА ШРИФТОВ	278
<i>Запрос доступных шрифтов</i>	278
<i>Генерирование имён шрифтов</i>	280
НАПИСАНИЕ ТЕКСТА В ПРЯМОУГОЛЬНОЙ ОБЛАСТИ	282
ИСПРАВЛЕНИЕ ПОВРЕЖДЕНИЙ В СЛУЧАЕ ПРОПОРЦИОНАЛЬНОГО ШРИФТА ТЕКСТА	285
ГЛАВА 20. ПЕЧАТЬ	288
КОНТЕКСТ ПЕЧАТИ	288
<i>Создание контекста печати</i>	289
<i>Модифицирование контекста печати</i>	289
ЗАПУСК ПРОЦЕССА ПЕЧАТИ	289
ПЕЧАТЬ ТРЕБУЕМЫХ ВИДЖЕТОВ	291
<i>Печать новой страницы</i>	291
<i>Печать скроллирующихся виджетов</i>	292
ПРИОСТАНОВКА И ВОЗОБНОВЛЕНИЕ РАБОТЫ ПЕЧАТИ	293
ЗАВЕРШЕНИЕ РАБОТЫ ПЕЧАТИ	294
ОСВОБОЖДЕНИЕ КОНТЕКСТА ПЕЧАТИ	294
ПРИМЕР	294
ГЛАВА 21. "ТАЩИТЬ И БРОСАТЬ"	297
МЕХАНИЗМ ТРАНСПОРТИРОВКИ	297
ИСПОЛЬЗОВАНИЕ "ТАЩИ И БРОСАЙ"	298
<i>Запуск операции "тащи и бросай"</i>	298
<i>Получение событий "тащи и бросай"</i>	300
<i>Отмена операции "тащи и бросай"</i>	302
РЕГИСТРАЦИЯ НОВЫХ ТРАНСПОРТНЫХ ТИПОВ	303
<i>Простая структура данных</i>	303
<i>Более сложная структура</i>	305
ТРАНСПОРТНЫЕ ФУНКЦИИ	308
ГЛАВА 22. РЕГИОНЫ	310
КООРДИНАТНОЕ ПРОСТРАНСТВО PHOTON!A	310
КООРДИНАТЫ РЕГИОНА	311
<i>Начало координат региона</i>	311
<i>Начальные размеры и расположение</i>	311
<i>О регионах потомка</i>	312
РЕГИОНЫ И ОТСЕЧЕНИЕ СОБЫТИЙ	313
МЕСТОРАСПОЛОЖЕНИЕ И ИЕРАРХИЯ	313
ИСПОЛЬЗОВАНИЕ РЕГИОНОВ	316
<i>Открытие региона</i>	316
<i>Размещение регионов</i>	317
СИСТЕМНАЯ ИНФОРМАЦИЯ	318
ГЛАВА 23. СОБЫТИЯ	319
СОБЫТИЯ МЫШИ	319
ГЕНЕРИРОВАНИЕ СОБЫТИЙ	321
<i>Нацеливание на определённые регионы</i>	322
<i>Нацеливание на определённые виджеты</i>	322
<i>События, генерируемые клавиатурными клавишами</i>	322
КООРДИНАТЫ СОБЫТИЯ	323
ОБРАБОТЧИКИ СОБЫТИЯ – НЕОБРАБОТАННЫЕ И ОТФИЛЬТРОВАННЫЕ ОТВЕТНЫЕ РЕАКЦИИ	324
НАКОПЛЕНИЕ СОБЫТИЙ	325
СЖАТИЕ СОБЫТИЙ	326
ПЕРЕТАСКИВАНИЕ	326
<i>Инициализация перетаскивания</i>	326
<i>Контурное перетаскивание</i>	327

Непрозрачное перетаскивание.....	328
Обработка событий перетаскивания.....	328
ГЛАВА 24. УПРАВЛЕНИЕ ОКНАМИ.....	331
ФЛАГИ УПРАВЛЕНИЯ ОКНАМИ.....	331
<i>Флаги отображения окна.....</i>	<i>331</i>
<i>Флаги управления окном.....</i>	<i>332</i>
<i>Оконные флаги уведомления.....</i>	<i>332</i>
ОТВЕТНАЯ РЕАКЦИЯ УВЕДОМЛЕНИЯ.....	333
ПОЛУЧЕНИЕ И УСТАНОВКА СОСТОЯНИЯ ОКНА.....	334
УПРАВЛЕНИЕ НЕСКОЛЬКИМИ ОКНАМИ.....	336
ФУНКЦИИ УПРАВЛЕНИЯ ОКНАМИ.....	336
ЗАПУСК САМОСТОЯТЕЛЬНОГО ПРИЛОЖЕНИЯ.....	336
МОДАЛЬНЫЕ ДИАЛОГИ.....	337
ГЛАВА 25. ПРОГРАММИРОВАНИЕ В PHOTON'E БЕЗ RHAV'A.....	341
ОСНОВНЫЕ ШАГИ.....	341
КОМПИЛИРОВАНИЕ И ЛИНКОВКА НЕ RHAV'ОВСКОГО ПРИЛОЖЕНИЯ.....	341
ОБРАЗЕЦ ПРИЛОЖЕНИЯ.....	342
<i>Что происходит.....</i>	<i>342</i>
ПОДСОЕДИНЕНИЕ ПРОГРАММНОГО КОДА ПРИЛОЖЕНИЯ К ВИДЖЕТУ.....	344
<i>Ответные реакции.....</i>	<i>345</i>
<i>Обработка событий.....</i>	<i>345</i>
ПОЛНЫЙ ПРИМЕР ПРИЛОЖЕНИЯ.....	345
ПРИЛОЖЕНИЕ 1. АРХИТЕКТУРА PHOTON'A.....	347
ПРОСТРАНСТВО СОБЫТИЙ.....	347
<i>Регионы и события.....</i>	<i>348</i>
СОБЫТИЯ.....	348
<i>Начальный набор прямоугольников.....</i>	<i>348</i>
<i>Накопленный набор прямоугольников.....</i>	<i>348</i>
РЕГИОНЫ.....	349
<i>Чувствительность.....</i>	<i>350</i>
<i>Непрозрачность.....</i>	<i>350</i>
<i>Краткая сводка атрибутов.....</i>	<i>350</i>
<i>Регистрация событий (event logging).....</i>	<i>350</i>
<i>Модификация событий.....</i>	<i>351</i>
<i>Взаимосвязь родитель/потомок.....</i>	<i>351</i>
<i>Координатное пространство Photon'a.....</i>	<i>351</i>
<i>Корневой регион.....</i>	<i>351</i>
ТИПЫ СОБЫТИЙ.....	351
КАК ВЛАДЕЛЬЦЫ РЕГИОНОВ УВЕДОМЛЯЮТСЯ О СОБЫТИЯХ.....	352
<i>Упорядоченный опрос.....</i>	<i>352</i>
<i>Синхронное уведомление.....</i>	<i>352</i>
<i>Асинхронное уведомление.....</i>	<i>352</i>
РЕГИОН УСТРОЙСТВ.....	353
<i>Фокусировка указателя.....</i>	<i>353</i>
<i>Фокусировка клавиатуры.....</i>	<i>353</i>
<i>События перетаскивания.....</i>	<i>353</i>
<i>Событие "тащи и бросай".....</i>	<i>353</i>
ДРАЙВЕРЫ ФОТОНА.....	354
<i>Драйверы ввода.....</i>	<i>354</i>
<i>Драйверы вывода.....</i>	<i>354</i>
ОКОННЫЙ МЕНЕДЖЕР PHOTON'A.....	355
<i>Регионы оконных рамок.....</i>	<i>356</i>
<i>Регион фокусировки.....</i>	<i>356</i>
<i>Регион рабочей области.....</i>	<i>356</i>
<i>Регион фона.....</i>	<i>356</i>
ПРИЛОЖЕНИЕ 2. ОБЗОР ВИДЖЕТОВ.....	357
ПРИЛОЖЕНИЕ 3. ПОДДЕРЖКА МНОГОЯЗЫЧНОСТИ UNICODE.....	360
ШИРОКИЕ И МНОГОБАЙТОВЫЕ СИМВОЛЫ.....	360

UNICODE.....	360
UTF-8 КОДИРОВАНИЕ	361
ФУНКЦИИ ПРЕОБРАЗОВАНИЯ	362
ДРУГИЕ КОДИРОВКИ	362
ДРАЙВЕРЫ КЛАВИАТУРЫ	363
СЛЕПЫЕ КЛАВИШИ (DEAD KEYS) И СКОМПОНОВАННЫЕ ПОСЛЕДОВАТЕЛЬНОСТИ	363
СКОМПОНОВАННЫЕ ПОСЛЕДОВАТЕЛЬНОСТИ PHOTON'A.....	364
ПРИЛОЖЕНИЕ 4. PHOTON ВО ВСТРОЕННЫХ СИСТЕМАХ	367
ПРИНИМАЕМЫЕ ДОПУЩЕНИЯ	367
ВВЕДЕНИЕ	367
ШАГ 1. ЭКСПОРТ ПЕРЕМЕННОЙ ОКРУЖЕНИЯ PHOTON_PATH	368
ШАГ 2. ЗАПУСК СЕРВЕРА PHOTON'A	369
ШАГ 3. ЗАПУСК ДРАЙВЕРОВ ВВОДА	369
ШАГ 4. ЗАПУСК МЕНЕДЖЕРА ШРИФТОВ	370
<i>Конфигурирование шрифтов</i>	<i>370</i>
<i>Запуск сервера шрифтов.....</i>	<i>371</i>
ШАГ 5. ПЕРЕКЛЮЧЕНИЕ В ГРАФИЧЕСКИЙ РЕЖИМ.....	372
<i>Установка карты в правильный режим.....</i>	<i>372</i>
ШАГ 6. ЗАПУСК ГРАФИЧЕСКОГО ДРАЙВЕРА.....	373
ШАГ 7. ЗАПУСК ОКОННОГО МЕНЕДЖЕРА	374
НЕОБХОДИМЫЕ ФАЙЛЫ	374
ШАГ 8. ЗАПУСК ВАШЕГО ПРИЛОЖЕНИЯ	374
НЕОБХОДИМЫЕ ФАЙЛЫ	374
ПОЯСНЕНИЯ.....	374
ФЛЭШ-ФАЙЛОВАЯ СИСТЕМА	375
ГРАФИКА.....	375
ПРИМЕР	376
<i>Требуемые бинарные файлы.....</i>	<i>376</i>
<i>Требующиеся библиотеки.....</i>	<i>377</i>
<i>Требуемые шрифты</i>	<i>378</i>
<i>Службы шрифтов.....</i>	<i>379</i>
<i>Сборка всего этого в единое целое</i>	<i>380</i>
ПОЛЕЗНЫЕ СОВЕТЫ.....	382
ПРИЛОЖЕНИЕ 5. ИСПОЛЬЗОВАНИЕ РНАВ ПОД MICROSOFT WINDOWS.....	383
PHOTON В ОДИНОЧНОМ ОКНЕ.....	383
ЗАВЕРШЕНИЕ РНАВ.....	383
ДОПОЛНИТЕЛЬНЫЕ ОПЦИИ	384
ФАЙЛОВЫЕ ИМЕНА С БУКВАМИ В ОБОИХ РЕГИСТРАХ	384
DDD – ОТЛАДЧИК ОТОБРАЖЕНИЯ ДАННЫХ	384
СТРОКА ЗАПУСКА ОТЛАДЧИКА	385
ФУНКЦИОНАЛЬНОСТЬ ПАНЕЛИ УПРАВЛЕНИЯ РЕСУРСАМИ	385
РАЗРАБОТКА ИНДИВИДУАЛЬНЫХ ВИДЖЕТОВ И РНАВ.....	386
<i>Статическое линкование Ваших индивидуальных виджетов.....</i>	<i>386</i>
ГЛОССАРИЙ	386
НЕНЕОБХОДИМОЕ ПОСЛЕСЛОВИЕ ПЕРЕВОДЧИКА.....	386

Небольшое пояснение переводчика

Это «Руководство программиста в Photon» представляет собой ну очень черновой вариант перевода, осуществлённый исключительно в ДСП-целях. Перевод писался, во-первых, в спешке, во-вторых, урывками. Поэтому читатель не найдёт здесь ни выверенных формулировок, ни корректного соответствия оригиналу, ни точности в терминах, ни... В общем, это просто “stub”, как выражаются в этой книге. Когда переводчик не понимал того, что написано, он не пытался вникать в суть, а, махнув рукой, писал что-то полувнятное и мчался дальше.

Кроме того, читатель вправе спросить, почему такие странные рисунки. Отвечаю: да, конечно, можно было просто скопировать в документ рисунки из оригинала, которые, в свою очередь, являются просто съёмками с экрана. Но. Такие рисунки плохо обрабатываются матричным принтером (счастливым обладателем какого является переводчик) – медленно и грязно. Поэтому оригинальные рисунки и заменены тем, что можно было бы назвать карикатурами, если бы не явное отсутствие таланта вышеупомянутого переводчика. Впрочем, мне кажется, произвести замену этих рисунков на оригинальные достаточно легко и быстро выполнимо. Было бы желание.

В заключение прошу прощения у всех тех, кому придётся читать это. Не надо ругать переводчика, ибо это даже не случай «не стреляйте в тапёра – он играет как может»: пресловутый переводчик совсем не может. Просто нужда заставила, соображения, что программистам, не владеющим английским, лучше прочитать это, чем вообще ничего.

Очень черновой перевод by Зайцев В.

От того, кто правил

Я постарался сделать этот текст более удобным для чтения как с экрана, так и в отпечатанном виде. Кроме того, где это было необходимо привел текст в соответствие с текущей редакцией на сайте QSSL. Для лучшего восприятия текста я все-таки заменил картинки в тексте на оригинальные – думаю так проще людям, впервые увидившим PhAB и иже с ним разобраться что к чему.

Горе-редактор Попелов М.

Введение

Итак, Вы уже, вероятно, увидели и испытали различные приложения Photon'a – оконный менеджер, просмотрщик Help'ов, игры и прочая – и готовы разрабатывать свои собственные приложения. Эта глава знакомит Вас с базовыми понятиями и концепциями, которые Вы будете использовать при разработке Photon'овских приложений.

В неё включены:

- обзор архитектуры Photon'a;
- построитель Photon'овских приложений – PhAB;
- концепции виджетов;
- образец программирования;
- библиотеки Photon'a;
- обзор построения приложений с помощью PhAB;
- написание приложений без PhAB.

Обзор архитектуры Photon'a

Менеджер Photon'a функционирует, как маленький сервер-процесс, включающий лишь несколько основных примитивов. Он создаёт трёхмерное пространство событий, заполненное областями и событиями. Этот менеджер не способен рисовать что-либо или работать с мышью, клавиатурой, пером.

Внешние, необязательные процессы – включая драйверы устройств, оконный менеджер или другие менеджеры – обеспечивают высокоуровневое функционирование оконной системы. Они связываются путём генерации событий в Photon'овском пространстве событий. Приложение Photon'a состоит из одной или более плоских, прямоугольных областей, которые действуют как "агенты" приложений в пространстве событий. Приложение рисует внутри области. Области сложены одна поверх другой в пространстве событий Photon'a. Область может иметь родительскую область и область одного с ней уровня (сёстры-области). Пользователь пребывает вонне пространства событий, наблюдая его спереди. Глубоко сзади в пространстве событий расположена особая область, называемая корневой областью (см. рис. 1).

Когда Вы запускаете приложение, Вы взаимодействуете с ним, и оно взаимодействует с другими приложениями и Photon'ом различным образом:

- Вы нажимаете клавиши и кнопки мыши;
- приложение выполняет графические операции;
- и т.д.

Эти взаимодействия называются событиями, и они перемещаются между областями в пространстве событий подобно фотонам света.

Пространство событий

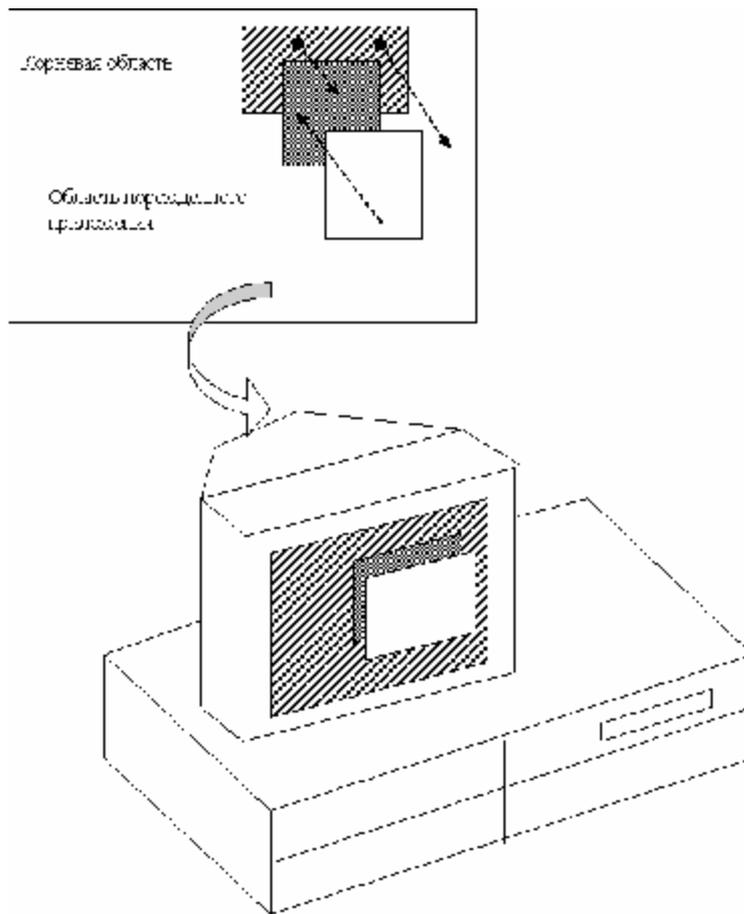


Рис. 1. Пространство событий Photon'a с точки зрения пользователя

Например:

Когда Вы нажимаете кнопку мыши, драйвер устройства генерирует событие и отправляет его назад через пространство событий (в сторону корневой области). Область, которая интересуется событием, может его перехватить и обработать, например, активизировав нажатие кнопки.

Когда Ваше приложение желает что-либо нарисовать, оно генерирует событие и отправляет его вперед (в сторону пользователя). Драйвер может перехватить событие и сформировать изображение на экране.

Каждая область может определить, в каких областях она заинтересована, путём установки своей чувствительности и непрозрачности:

Область, чувствительная к определённому типу событий, уведомляет приложение всякий раз, когда такое событие пройдёт сквозь неё.

Область, непрозрачная к определённому типу событий, блокирует его путём отсечения своей собственной зоны от области событий.

Для получения более полной информации см. приложение к Архитектуре Photon'a.

Построитель Photon'овских приложений – PhAB

microGUI (микроГрафический Пользовательский Интерфейс) включает очень мощный инструмент разработки, называемый Построитель Приложений Photon'a (Photon Application Builder,

сокращённо PhAB или appBuilder). Это инструмент визуального проектирования, генерирующий базовый код C и/или C++, обеспечивающий пользовательский интерфейс для Ваших приложений. С помощью PhAB Вы можете в чрезвычайной степени уменьшить объём программирования, требуемый для построения Вашего приложения. Вы можете сэкономить не только время, необходимое для написания части Вашего кода, относящегося к пользовательскому интерфейсу, но также время на отладку и тестирование. PhAB поможет Вам получить Ваши приложения более быстро продаваемыми и с более профессиональными результатами.

PhAB берёт на себя задачу о проектировании и создании окон, меню, диалогов, иконок, виджетов (кнопок, меток и прочая), и обратной реакции виджетов со многими расширениями. PhAB даёт Вам возможность получить доступ и создавать модули PhAB внутри Вашего собственного кода. Он также предоставляет набор функций-утилит для установки базы данных виджетов, которые Вы сможете использовать многократно столько раз, сколько потребуется, а не создавать виджеты с нуля.

Получение немедленных результатов

PhAB позволяет Вам избежать процесса создания пользовательского интерфейса вручную методом проб и ошибок. Вместо написания кода для каждой кнопки, окна и иных виджетов, Вы создаёте желаемые вам виджеты просто "указав и кликнув"¹.

Как только вы создали виджет, PhAB отображает его на экране вместе со всеми ресурсами, управляющими тем, как данный виджет выглядит и ведёт себя. Изменить любой ресурс виджета легко – просто щёлкните на ресурсе, выберите новое значение, и дело сделано. Так же легко переместить виджет или изменить его размеры – просто укажите на виджет и перетащите его мышкой.

Концентрация на функциональности

Подобно другим средам разработки GUI (Графическим Пользовательским Интерфейсам), PhAB позволяет Вам прикрепить коды функций к ответным реакциям виджетов (callback – ответная реакция в вольном переводе, это термин PhAB, особая вкладка в PhAB. Прим. пер.), обеспечивая Вашему приложению функциональность. Например, Вы можете прикрепить код функции к кнопке, так что функция будет вызываться каждый раз, когда пользователь щёлкнет на кнопке.

Далее, PhAB не принуждает Вас писать и прикреплять код, необходимый для "склеивания" вместе различных частей Вашего интерфейса. Вместо этого Вы можете прикрепить ответные реакции виджетов непосредственно к любому окну, диалогу или меню. Единственный код, о котором Вам придётся беспокоиться – это код, специфический для Вашего приложения.

Создание прототипов без написания кода

Как только Вы завершили любую часть пользовательского интерфейса, Вы можете, имея PhAB, сгенерировать весь C и/или C++ код, требуемый для того, чтобы оживить интерфейс, что означает, что Вы можете создать полностью прототип, не написав ни строки кода.

После того, как Вы сгенерировали и откомпилировали код, Вы можете запустить прототип на выполнение, чтобы посмотреть, как интерфейс функционирует. Например, если Вы связали с диалогом кнопку, щелчок на кнопке приведёт к всплыванию диалогового окна. Вы немедленно почувствуете, как интерфейс будет "ощущать" пользователя. Фактически PhAB делает процесс

¹ Прим.: "point and click" – дословно "указать и выбрать мышью" – термин, означающий метод взаимодействия с интерфейсом визуальных средств разработки приложений, заключающийся в том, что для создания нового экземпляра объекта надо щёлкнуть на нём и щёлкнуть там, куда его поместить.

построения и тестирования настолько эффективным, что Вы можете даже сидеть с Вашими пользователями и проектировать прототипы вместе.

После завершения проектирования прототипа, Вы можете встроить его в Ваше действующее приложение. Или Вы можете в любой момент остановить разработку прототипа, написать какие-то функции обратной связи, поэкспериментировать, как они работают, и затем вернуться к проектированию прототипа. Вы всегда можете провести тонкую регулировку всех аспектов Вашего приложения и делать всё, что пожелаете, пока приложение работает.

Сокращение размера кода

Вашему приложению может понадобиться использовать в различных частях его интерфейса одни и те же виджеты. С PhAB Вам не понадобится устанавливать их каждый раз, когда в этом появится нужда. Вместо этого Вы однажды определяете виджеты, помещаете их в базу данных виджетов и затем, используя функции языка C, обеспечиваемые PhAB, используете эти виджеты, когда Вам понадобится. Применяя такой подход, Вы можете уменьшить размер кода, требуемого для создания виджета, до одной строки.

Создание последовательных приложений

С PhAB Вам редко понадобится разрабатывать приложение с нуля. Например, если Вы уже создали окна и диалоги в существующем приложении, вы можете свободно перебросить их в новое приложение. Вы можете также создать центральную базу данных виджетов, чтобы импортировать их во все Ваши приложения для создания единообразного вида и реагирования.

Создание всех разновидностей приложений

С PhAB вы можете ускорить разработку без ухудшения функциональности. Вы можете создавать все разновидности приложений. Например, мы использовали PhAB для разработки почти всех приложений, поставляемых с Photon'ом, включая просмотрщик файлов помощи (Helpviewer), терминал (Terminal application), менеджер рабочего стола (Desktop Manager), копировщик экрана (Snapshot), все игры и “демки” – даже, собственно, сам PhAB!

Лучшим введением в PhAB является использование его, так что начинайте работать, используя руководства. Через очень короткое время Вы будете способны собрать воедино очень специализированные прототипы. Когда Вы будете готовы начать программирование своего приложения, Вы сможете прочитать разделы, имеющие отношение к тем виджетам, которые Вы захотите использовать. Мы снабдили полными исходными кодами все “демки” и игры, созданные с помощью PhAB. Вы можете загрузить любой из них в PhAB и изучить для использования идей в Ваших собственных приложениях. Хорошим примером, использующим большую часть возможностей PhAB, является приложение ВидеоПокер.

Концепции виджетов

При создании нового пользовательского интерфейса (UI), проще собрать интерфейс из набора стандартных компонентов, таких как ползунки, списки, меню и кнопки, нежели реализовывать каждый из элементов UI с нуля. Каждый стандартный компонент, включённый в UI, является объектом, называемым виджетом.

Виджеты Photon'a обеспечивают набор компонентов UI, в большей или меньшей степени согласующихся с другими оконными системами, которые Вы могли видеть.

Набор виджетов создан на объектно-ориентированной основе, нестрого говоря, базирующейся на библиотеке встроенного инструментария X-windows (X Toolkit Intrinsic library, Xt). Если вы уже знакомы с Xt, Вы увидите, что здесь применены во многом те же концепции.

Виджет объединяет данные и действия, требуемые для обеспечения работоспособности данного элемента UI. Подобное группирование в объекте данных и действий называется инкапсуляцией.

Виджет инкапсулирует в себе знание о том, как:

- нарисовать себя;
- отзываться на генерируемые пользователем события (например, нажатие левой кнопки мыши);
- восстановить себя, перерисовав, когда он окажется повреждённым (например, когда перекрывающее его окно будет закрыто).

Кроме того, имеется несколько виджетов, называемых контейнерами, содержащие другие виджеты и управляющие их компоновкой.

Виджет также скрывает подробности того, как он выполняет свои обязанности перед внешним миром. Этот принцип, известный как сокрытие информации (information hiding), отделяет внутреннюю реализацию виджета от его общедоступного интерфейса.

Общедоступный интерфейс состоит из всех атрибутов, видимых другими объектами, а также действий, которые другие объекты могут совершать над этим виджетом. Атрибуты в общедоступном интерфейсе виджета называются ресурсами.

☞ Плюсом для Вас, как для программиста, является то, что Вы, используя виджет, не знаете деталей его внутренней реализации – Вам необходимо просто знать его общедоступный интерфейс, как виджет создать и уничтожить, и как обращаться с его ресурсами.

Не каждый объект является уникальным. Объекты, исполняющие те же самые функции и имеющие одинаковый общедоступный интерфейс, относятся к одному классу. Виджеты, обеспечивающие один и тот же компонент пользовательского интерфейса, принадлежат к одному классу виджетов. Методы класса окон обеспечивают общую функциональность класса.

Несколько классов виджетов могут иметь общие атрибуты и действия. В этих случаях эти классы виджетов могут быть выделены в категорию subclasses одного суперкласса, или родительского класса. Атрибуты и действия инкапсулированы в суперклассе; subclasses наследуют их от родительского класса. Считается, что сами subclasses наследуются из родительского класса. Библиотека Photon'a позволяет “виджетному” классу наследоваться только от одного “виджетного” класса. Такая взаимосвязь известна, как одиночное наследование. Взаимосвязи между классами виджетов могут быть изображены как дерево, известное как иерархия классов.

С помощью браузера (если это – документ HTML) Вы можете выбрать виджет на диаграмме (см. рис. 2).

Вложенность экземпляров виджетов в GUI Вашего приложения создаёт иную иерархию виджетов. Чтобы отличить её от иерархии классов, её называют семейством виджетов.

Библиотека виджетов Photon'ов функционирует как фабрика виджетов. Она предоставляет набор функций, которые позволяют программисту создать новый виджет определённого “виджетного” класса и затем управлять этим виджетом. Однажды созданный, виджет имеет все характеристики “виджетного” класса. Будучи унаследованным, он также имеет все характеристики суперкласса своего “виджетного” класса.

Новый виджет является экземпляром “виджетного” класса. Создание нового виджета определённого класса называется также инициализацией (instantiating) виджета. Этот термин не вполне точен, потому что Вы на самом деле инициализируете “виджетный” класс. Это отражает базовую тенденцию данного руководства – ссылаться и на виджеты и на “виджетные” классы, просто как на виджеты.

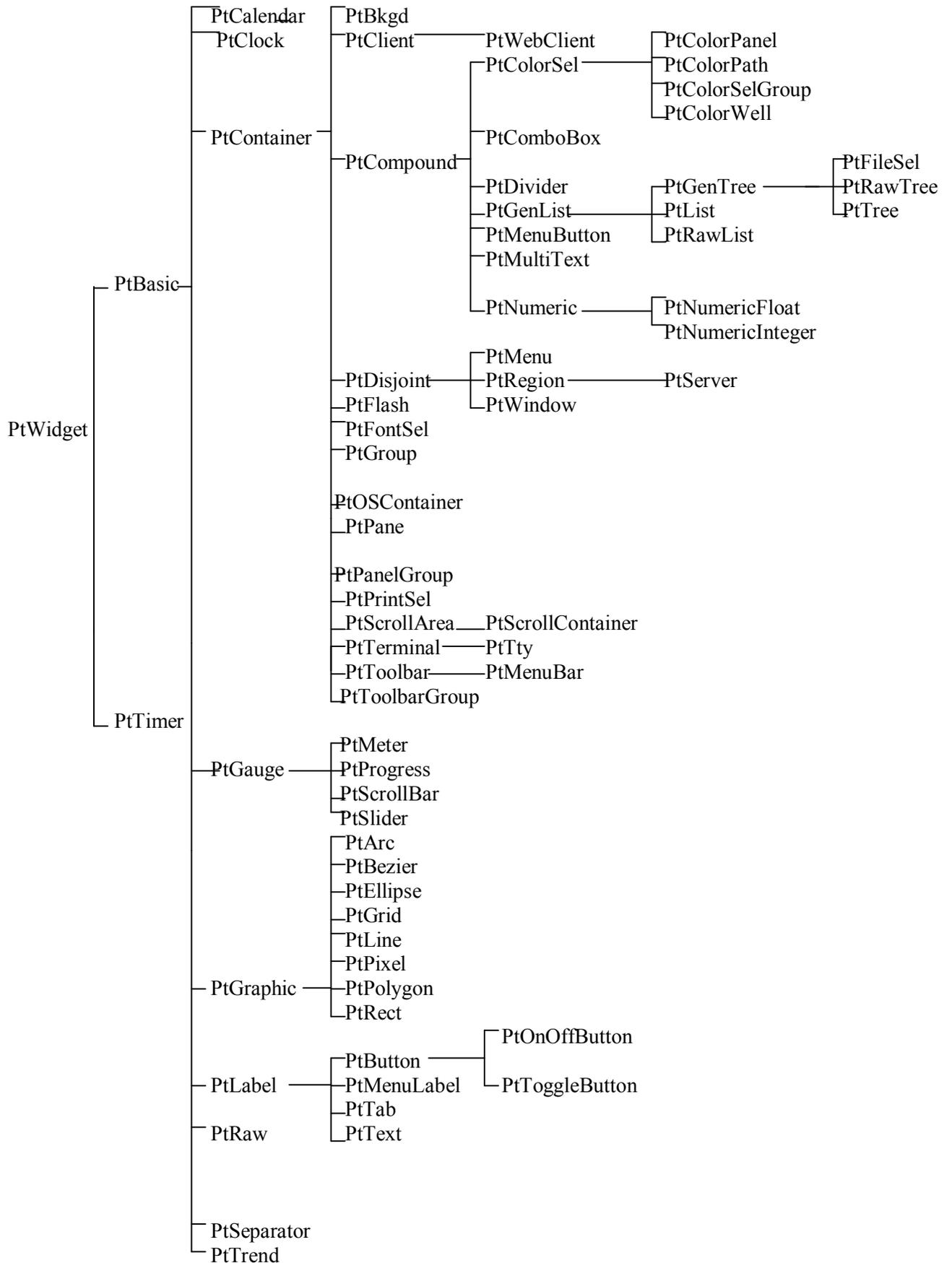


Рис. 2. Иерархия виджетов Photon'a

Ресурсы виджетов используются для конфигурирования того, как они выглядят на экране, и их поведения. Вы можете редактировать ресурсы в PhAB, и после создания виджета Вы можете изменить многие из них вызовами функций `PtSetResource()` или `PtSetResources()`. Ресурсы широко используются для управления данными, отображаемыми виджетом и задания того, как их отображать.

Например:

ресурс `Pt_ARG_TEXT_STRING` виджета `PtLabel` является строкой, выводимой на экран; ресурсы виджета `PtButton` определяют, отображает ли кнопка строку и/или картинку (`image`), её текст, картинку, цвет, и что происходит, когда пользователь выбирает кнопку.

Важным типом ресурса, предоставляемого виджетами, является список ответных реакций (`callback list`), являющийся списком функций, запускаемых виджетом в ответ на определённые значимые пользовательские события. Например, виджет текстовой области вызывает соответствующие функции из своего списка ответных реакций всякий раз, когда пользователь вводит новое значение и нажимает `Enter`. При разработке приложения Вы можете добавить реакции в список ответных реакций виджета, задавая соответствующие действия в ответ на пользовательские события.

Жизненный цикл виджета

Виджет имеет присущий ему жизненный цикл, как показано ниже.

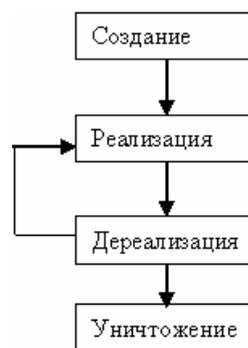


Рис. 3. Жизненный цикл виджета

Когда виджет оказывается востребованным, он создаётся или инициализируется (`instantiated`). После его создания появляется возможность манипулировать его атрибутами, или над ним можно производить действие.

Когда виджет создан, он не становится немедленно видим в пользовательском интерфейсе. Он должен быть реализован (`realized`). Если Вы используете PhAB, Ваши виджеты реализуются автоматически; если Вы PhAB не используете, Вы должны реализовать их, используя функцию `PtRealizeWidget()`.

Реализация виджета автоматически реализует всех его потомков. Photon гарантирует, что потомки виджета реализуются перед ним самим, так что виджет может вычислить размер своей инициализации, основываясь на размерах своих детей. Вы можете задать, чтобы приложение уведомлялось, что виджет реализован, путём регистрации ответной реакции в списке реакций `Pt_CB_REALIZED`. С помощью функции `PtUnrealizeWidget()` Вы можете временно скрыть виджет от пользовательского интерфейса, дереализовав его. Что касается реализации, Вы можете уведомить приложение, используя ресурс ответной реакции `Pt_CB_UNREALIZED`.

Когда виджет оказывается больше не нужным, Вы можете убить его. Вы можете уничтожить виджет, вызвав функцию `PtDestroyWidget()`. В действительности виджет не уничтожается немедленно – он маркируется как подлежащий удалению в соответствующее время и добавляется в список виджетов, подлежащих уничтожению. Эти виджеты обычно уничтожаются внутри

главной петли приложения после того, как обработают все ответные реакции, привязанные к событию.

Ваше приложение может определить ответные реакции `Pt_CB_DESTROYED` для любого виджета. Эти ответные реакции вызываются, когда виджет маркируется, как подлежащий уничтожению. Вы можете задать, чтобы приложение уведомлялось, когда виджет в действительности уничтожен, регистрацией функции со списком ответной реакции на уничтожение (`Pt_CB_IS_DESTROYED`) для виджета. Это особенно полезно для очистки структур данных, связанных с виджетом.

Геометрия виджета

Вы можете рассматривать виджет, как картинку или, подмонтированную фотографию. Виджет закрепляется на раме, называемой рамкой (*by a frame, called a border – велик и могуч английский язык. Прим. пер.*). Для виджета рамка – это набор контуров и создающие эффект выпуклости фаски, которые могут быть нарисованы вокруг внешних сторон.

Часть виджета, используемая для рисования, называется канвой (canvas). Для `PtWidget` – это область внутри рамки виджета. Для `PtBasic` и его потомков канва – это область внутри рамки виджета и границ. Другие виджеты, такие как `PtLabel`, определяют другие границы. Границы, формирующие затемнение (matt) и затемняющие любую часть канвы, распространяются за пределы отсечённой части. Эта затемнённая область иногда называется как отсечённая (clipping) область (см. диаграмму на рис. 4).

☞ Для наглядности на диаграмме канва и границы показаны различным цветом. В реальном виджете они одного цвета.

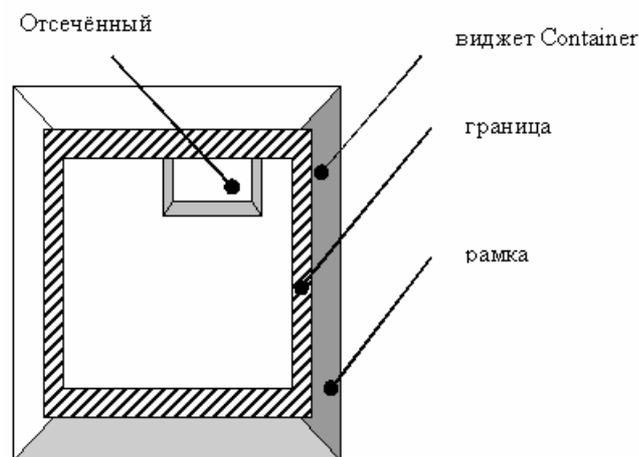


Рис. 4. Анатомия виджета PtBasic

Для виджета рамка является необязательной. Она рисуется, только если виджет подсвечен (т.е. в его ресурсе `Pt_ARG_FLAGS` установлен флаг `Pt_HIGHLIGHTED`). Рамка состоит из различных необязательных компонентов, в зависимости от установок в ресурсе `Pt_ARG_BASIC_FLAGS`.

Компонентами, рассматривая их от самых наружных вовнутрь, являются:

- однопиксельная линия "гравировки";
- однопиксельная наружная контурная линия;
- фаска, ширина которой установлена в `Pt_ARG_BEVEL_WIDTH`;
- однопиксельная внутренняя контурная линия.

Виджет имеет несколько важных атрибутов, определяющих геометрию этих элементов. Размеры виджета, `Pt_ARG_DIM` – это общий размер виджета, включая его границы.

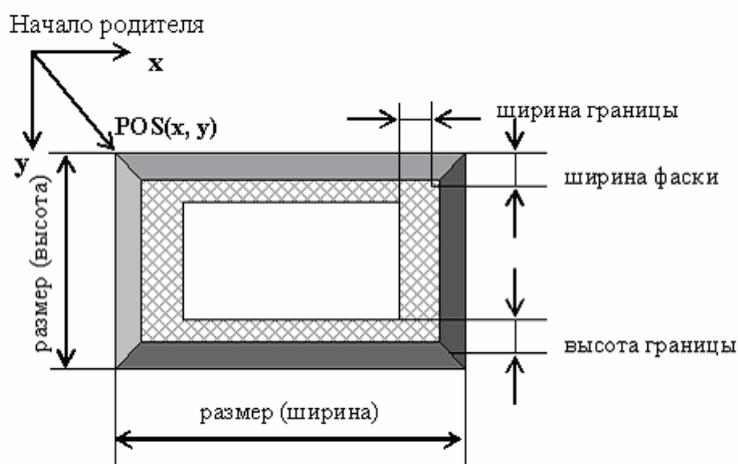


Рис. 5. Позиция и размеры виджета

`Pt_ARG_MARGIN_WIDTH` определяет ширину границы слева и справа от канвы, и `Pt_ARG_MARGIN_HEIGHT` определяет высоту границы над и под канвой. Эти ресурсы определены в `PtBasic`. Другие классы виджетов определяют свои собственные ресурсы границ, которые могут быть добавлены к ширине и высоте базовой границы. Например, виджет надписи (label widget) обеспечивает различные границы для левой, правой, верхней и нижней сторон виджета. Они добавляются к базовым ширине и высоте для определения объёма пространства, оставляемого на каждую сторону канвы. Начальной точкой виджета (в целях выполнения любой прорисовки или позиционирования любого порождаемого виджета) является верхний левый угол канвы. Все координаты, определяемые для виджета, являются относительными этого начала, как координаты всех событий, получаемых виджетом. Например, если виджет является контейнером, позиции всех его "детей" являются относительными этой точки.

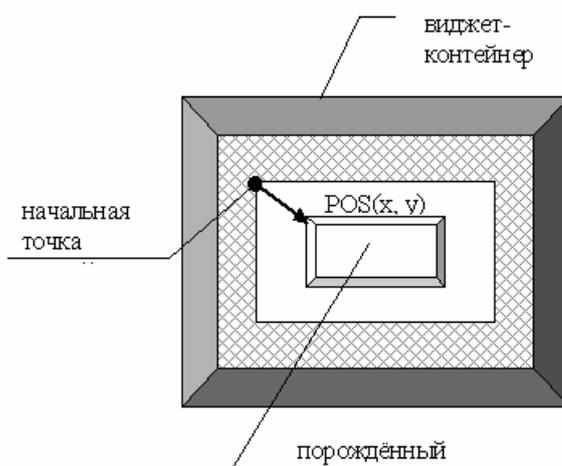


Рис. 6. Начальная точка виджета и позиция его "ребёнка"

Для позиционирования "детей" контейнеры описаны только внешним краем рамки виджета. Позиция виджета содержится в ресурсе `Pt_ARG_POS`. Эта позиция является точкой в верхнем левом углу внешнего контура рамки виджета. Контейнер позиционирует своих детей выравниванием по этому ресурсу.

Доступ и модификация позиции и размеров виджета могут быть достигнуты одновременно использованием ресурса `Pt_ARG_AREA`, предоставляемого виджетом. Пространство виджета – это прямоугольник, определяемый позицией виджета и его размерами. Обычно оно не может быть вычислено до того, как виджет не будет реализован; Вы можете принудить виджет вычислить своё пространство вызовом `PtExtentWidget()`; чтобы вынудить виджет и все его порождения вычислить свои пространства, необходимо вызвать `PtExtentWidgetFamily()`. Как только пространство вычислено, Вы можете узнать его, получив данные ресурса `Pt_ARG_EXTENT`, или вызовом `PtWidgetExtent()`.

Парадигма (система понятий) программирования

Давайте сравним, как Вы пишете не-PhAB (Photon) приложения текстового режима, и приложения PhAB.

Приложение текстового режима

Когда Вы пишете не-Photon'ное (текстового режима) приложение, Вы в основном концентрируетесь на основной программе, из которой Вы делаете такие вещи, как:

- инициализация приложения
- установка обработчиков сигналов
- посылка и получение сообщений
- итерации
- вызов подпрограмм, если требуется
- связь с консолью
- и, наконец, выход.

Основная программа

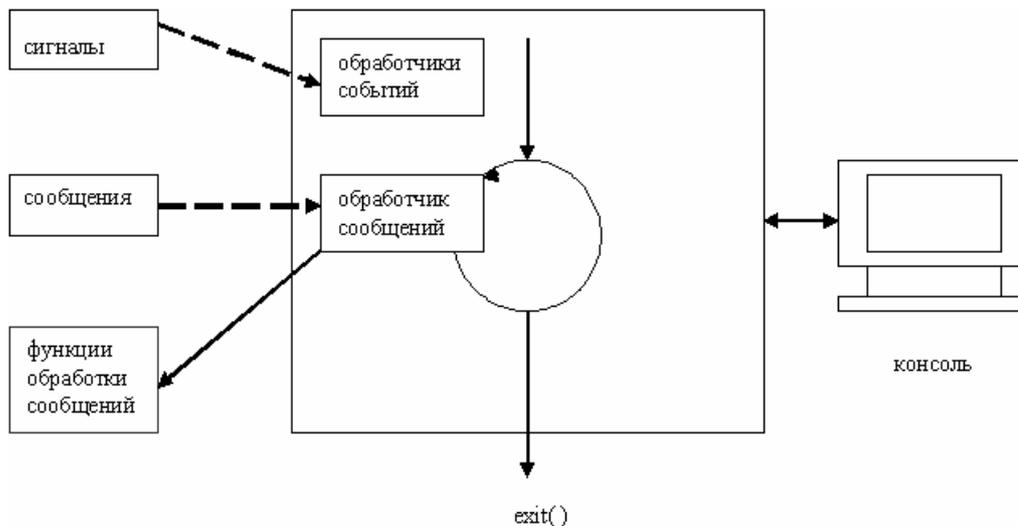


Рис. 7. Структура приложения текстового режима

Не-PhAB приложение

Приложение Photon'a, написанное без PhAB, похоже на приложение текстового режима, за исключением того, что вы также:

- инициализируете и реализуете виджеты приложения;
- устанавливаете ресурсы виджетов, включая такие:
 - размер и позиция
 - привязка
 - текст
 - список ответных реакций
 - прочая
- пишете подпрограммы ответных реакций для обработки событий виджетов. При этом Вам может понадобится:
 - создать окна и их виджеты, установить ресурсы, и затем реализовать их
 - создать меню из виджета PtMenuButton, установить ресурсы и ответные реакции, реализовать эти меню
 - уничтожить виджеты
 - прочая
- вызвать PtMainLoop() в Вашей основной программе для обработки событий.

Обычно одна из Ваших ответных реакций выполняет завершение приложения. Написание приложений без PhAB означает, что Вы будете работать непосредственно с виджетами – со всей их кучей.

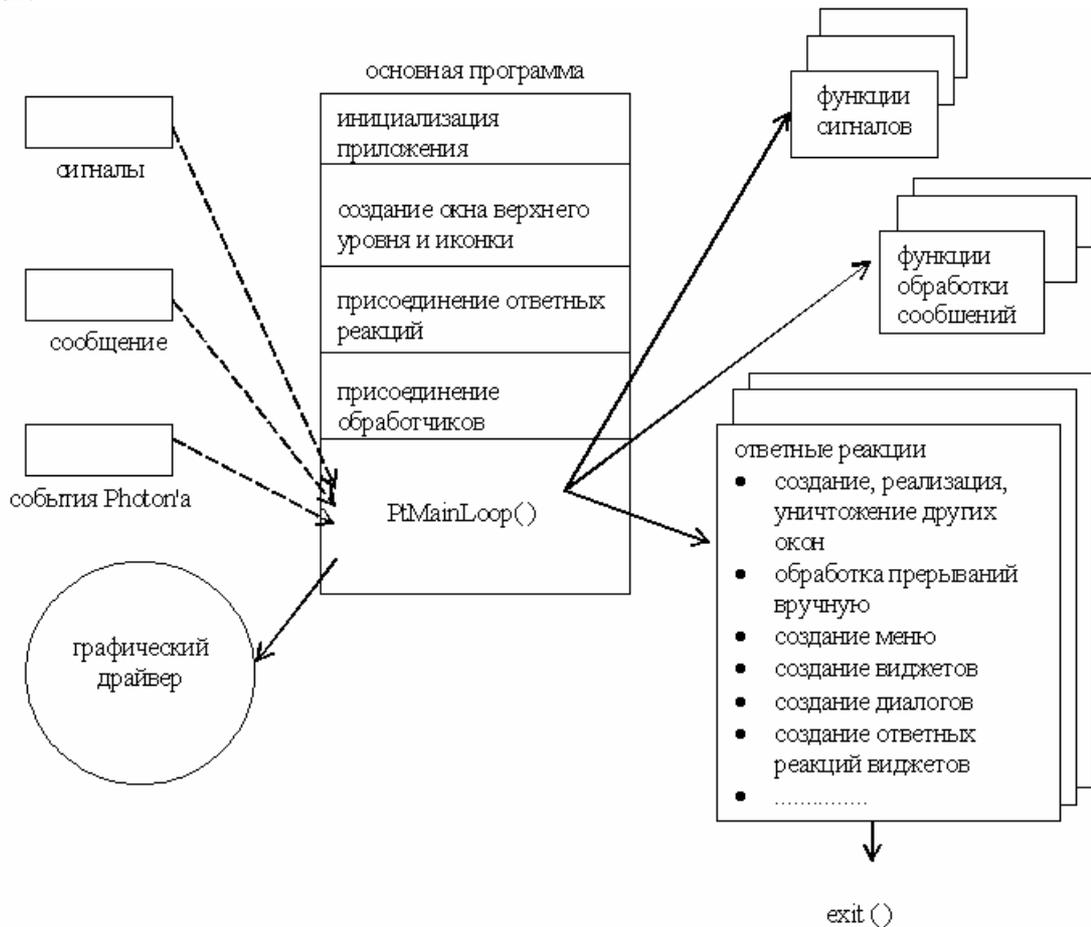


Рис. 8. Структура приложения Photon, написанная без использования PhAB

PhAB приложение

Когда Вы разрабатываете PhAB приложение, main-программа Вам предоставляется. Вместо того, чтобы заботиться об основной программе, Вы:

- обеспечиваете функцию, которая инициализирует приложение;
- устанавливаете обработчики сигналов, которые обрабатывают сигналы, когда те прибывают, и вызываете написанные Вами функции работы с сигналами;
- устанавливаете функции ввода для сообщений;
- пишете ответные функции для обработки событий от виджетов.

Основная программа всегда закольцована, обрабатывая события, когда они случаются. Обычно одна из Ваших ответных функций завершает приложение. PhAB обрабатывает для Вас массу деталей – Вы концентрируетесь на функциональности Вашего приложения, а не на виджетах.

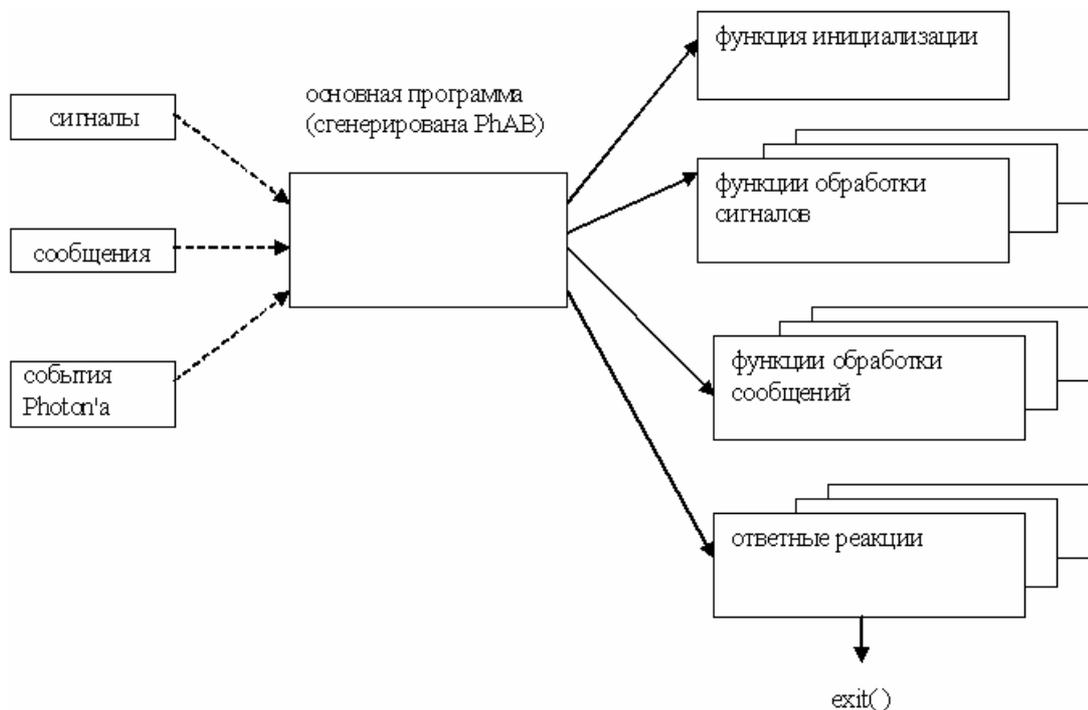


Рис. 9. Структура приложения Photon'a, написанная PhAB

Дополнительно, Вы не имеете в Вашем коде размер и позиции виджетов; Вы делаете это визуально в PhAB. PhAB также просматривается после инициализации, реализации, дереализации и уничтожения Ваших виджетов. PhAB даже обеспечивает модулем меню, что делает простым создание меню. Вы можете видеть, почему мы рекомендуем использовать PhAB!

Библиотеки Photon'a

Интерфейс программирования приложений Photon'a (API) – организован как набор функций, каждая из которых характеризуется двухсимвольным префиксом:

- AI** функции перевода PhAB (PhAB Translation functions), позволяющие Вам работать с файлами перевода (translation files) (для приложений PhAB или баз данных сообщений) без использования редактора перевода. Эти подпрограммы отсутствуют в библиотеке совместного доступа, чтобы их использовать, необходимо линковать приложение с библиотекой phexlib;

Ar	функции PhAB, работающие с модулями, базами данных виджетов, переводом (translation) и прочая. Эти подпрограммы отсутствуют в библиотеке совместного доступа, чтобы их использовать, необходимо линковать приложение с библиотекой Ar;
mbstr	функции строк многобайтных символов. См. приложение "Поддержка многоязычности Unicode";
Pd	функции работы с рисуемым контекстом;
Pf	службы шрифтов, включая метрики текстов, и генерация побитых карт символьных строк (generation of bitmaps of character strings). Для более полной информации см. главу "Шрифты";
Pg	низкоуровневые графические функции, позволяющие получить доступ к богатому набору примитивов в графических драйверах. Эти функции используются в библиотеках виджетов и могут также быть вызваны непосредственно, используя виджет PtRaw. См. главу "Необработанное рисование и мультипликация";
Ph	примитивы Photon'a, являющиеся совокупностью запросов на рисование и отправляющие их в микроядро Photon'a для управления и отсекающего до тех пор, пока они не достигнут графического драйвера, готовые к отображению на экран. Поскольку эти функции не являются общеиспользуемыми при программировании приложений, их тяжело использовать в библиотеках графики и виджетов;
Pi	функции работы с образами (image). См. раздел "Работа с образами" в главе "Необработанное рисование и мультипликация";
Pm	функции работы с памятью, которые можно использовать для уменьшения мерцания. См. раздел "Мультипликация" в главе "Необработанное рисование и мультипликация";
Pp	функции печати, позволяющие устанавливать режимы и управлять печатью. См. главу "Печать";
Pt	функции набора инструментов виджетов для создания, реализации и уничтожения виджетов, получения и установки ресурсов и прочая. Кроме использования виджетов в библиотеке виджетов Photon'a, Вы можете использовать виджеты третьей стороны или свои собственные виджеты;
Px	функции расширений, работающие с загружаемыми образами, с файлами конфигурации, и другие полезные подпрограммы. Эти подпрограммы отсутствуют в библиотеке совместного доступа, чтобы их использовать, необходимо линковать приложения с библиотекой phexlib;
Rt	функции таймера реального времени. См. раздел "Таймеры" в главе "Работа с кодом";
wc	функции работы со строками символов расширенного (16-битного) набора. См. приложение "Поддержка многоязычности Unicode".

Функции и структуры данных этих библиотек описаны в книге "Справочник по библиотеке Photon'a". Функции mbstr, Pd, Pf, Pg, Ph, Pi, Pm, Pp, Pt, Rt и wc находятся в главной библиотеке Photon'a. Функции, используемые для растеризации потока рисования Photon'a, находятся в библиотеке phrender. Библиотеки ph, phrender и Ar доступны в форме общего доступа и в статической.

Вы можете линковать Ваше приложение с библиотеками общего доступа, делая Ваше приложение меньшим по размеру. Для более полной информации, см. раздел "Выбор библиотек" в главе "Генерирование, компилирование и запуск кода".

☞ Библиотека общего доступа ph не включает ничего, что требует операций с плавающей запятой (кроме виджета PtNumericFloat). Статическая версия это делает.

Функции Al и Px включены в библиотеку расширения phexlib, которая доступна только в статической форме.

- ! Библиотека `photon` предназначена только для приложений, созданных под микро-GUI Photon'a версии 1.14. Не комбинируйте эту библиотеку с текущими библиотеками или хедер-файлами, в противном случае Ваше приложение не будет работать правильно.

Обзор построения приложений под PhAB

Шаг 1. Создание модулей

Для создания приложения пользовательского интерфейса в PhAB, Вы начинаете работу с первичными строительными блоками, называемыми модулями. Модули выглядят и функционируют по большей части так, как Вы видите в большинстве приложений Photon'a. Вы можете проектировать пользовательский интерфейс просто с одним модулем, но в большинстве приложений Вы, вероятно, будете использовать несколько модулей и назначать каждому свою роль. Как правило, каждый модуль группирует вместе взаимосвязанную информацию и позволяет пользователю особым образом взаимодействовать с этой информацией. Чтобы помочь Вам управлять требованиями виртуальности любого приложения, PhAB предоставляет несколько типов модулей:

- `window` – обычно используется для главной деятельности приложений. Типичное приложение имеет одно основное окно, которое открывается, когда окно стартует.
- `dialog` – позволяет приложению обмениваться информацией с пользователем.
- `menu` – предоставляет пользователю команды.
- `icon` – определяет иконку для приложения.
- `picture` – может быть использовано различным образом. Например, Вы можете использовать картинку для обеспечения удобной базы данных виджетов или для изменения содержания существующего модуля.

Для получения более полной информации, см. главу "Работа с модулями".

Шаг 2. Добавление виджетов

После того, как вы создали модуль, Вы готовы разместить на нём виджеты. Для добавления виджета просто щёлкните на желаемой иконке в палитре виджетов PhAB, затем щёлкните там, где Вы хотите его поместить. PhAB даёт Вам возможность добавить любой виджет, пришедший с окружением разработки Photon. Вы можете назначить для виджета:

- отображаемые или редактируемые величины – примерные включённые надписи (*examples include labels*), текст или многострочный текст;
- наличие выбора – примерные включённые списки (*examples include lists*), комбинированные управляющие элементы (*comboboxes*), и группы;
- отображение графики – примерные включённые побитовые карты, образы, линии, прямоугольники, эллипсы и многоугольники;
- отображение областей скроллинга – примерные включённые линейки прокрутки и контейнеры скроллинга;
- инициализирующие действия – примерные включённые кнопки, содержащие текст или образы.

Для задания того, как виджет выглядит и функционирует, Вы устанавливаете его атрибуты или ресурсы. Панели управления и Редакторы ресурсов PhAB'a делают это простым. Просто щёлкните на ресурсе, который Вы желаете изменить, и затем выберите или введите новое значение.

Вы можете определить событие виджету (*you can event customize*) и затем сохранить его, как шаблон для использования при создании подобных виджетов.

Для более полной информации см. главу "Редактирование ресурсов и ответных реакций в PhAB".

Шаг 3. Прикрепление ответных реакций

Вы создали свои модули и разместили на них виджеты. Теперь Вы готовы определить, как приложение должно работать. Чтобы сделать это, Вы используете ответные реакции.

Каждый виджет Photon'a поддерживает несколько типов ответных связей. Для прикрепления функций, написанных на коде (code functions) к ответной реакции, Вы устанавливаете ресурс или используете предусмотренную удобную функцию. Каждый раз, когда встречается состояние ответной реакции, виджет выполняет функцию на коде.

В PhAB Вы свободны концентрироваться в Ваших ответных реакциях на написание кода, специфического для приложения – Вы не придётся создавать код для "склеивания" интерфейсных компонентов вместе, потому что PhAB обеспечивает уровень ответных реакций, называемый линкованием ответных реакций (link callback). Используя линкование ответных реакций, Вы можете прикрепить ресурс ответной реакции виджета непосредственно к окнам, диалогам, меню и многим другим вещам, кроме кода приложения.

Линкование ответных реакций также позволяет Вам добавить функциональность, которая не была доступна, когда Вы прикрепляли ответные реакции "вручную". Например, если Вы подлинковали диалог к виджету кнопки, Вы можете задать, где появляется диалог. Вы можете также задать установочную функцию, которая будет автоматически вызываться перед реализацией диалога, после его реализации, либо и так и сяк.

Расширенная функциональность, обеспечиваемая подлинковыванием ответных реакций, есть простейший способ конструирования пользовательского интерфейса. Фактически Вы можете создать прототип целого приложения без необходимости написания какого-либо кода. Для получения более полной информации см. главу "Редактирование ресурсов и ответных реакций в PhAB".

Шаг 4. Генерация кода

Вы создали модули своего приложения и создали подлинкованные ответные реакции для склеивания вместе различных компонентов. Теперь Вы готовы сгенерировать и откомпилировать код для запуска Вашего проекта приложения в исполняемом виде.

Чтобы сделать это, Вы открываете диалог "Build+Run" и просто щёлкаете несколько кнопок. Вот так. Этот диалог включает файловый менеджер, так что Вы можете также редактировать исходный код и манипулировать файлами – без необходимости выхода из окружения PhAB.

Для получения более полной информации см. главу "Генерация, компиляция и запуск кода на выполнение".

Шаг 5. Запуск Вашего приложения на выполнение

После того, как Вы сгенерировали, откомпилировали и отлинковали Ваше приложение, Вы можете запустить его на выполнение прямо из диалога "Build+Run". Используя этот же самый диалог, Вы можете даже запустить Ваше приложение под отладчиком для плавной отладки.

Для получения более полной информации см. главу "Генерация, компиляция и запуск кода на выполнение".

Шаг 6. Повторение любого предыдущего шага

После того, как Вы сгенерировали и откомпилировали Ваше приложение, Вы можете изменить интерфейс, прикрепить ответные реакции и регенерировать код столько раз, сколько нужно.

Написание приложений без PhAB

Мы рекомендуем при разработке Вашего приложения использовать PhAB. Однако, если Вы не планируете использовать PhAB, Вам стоит прочесть полностью это руководство (особенно главу "Программирование Photon'a без PhAB"), чтобы познакомиться со всеми фундаментальными основами Photon'a перед тем, как начать создание приложения. Затем стоит просмотреть "Справочник виджетов".

Глава 1. Уроки

Лучшим способом получить знания о PhAB – это использовать его. Эта глава представляет из себя практические уроки, позволяющие Вам быстро начать создавать приложения:

- Перед тем как Вам начать...
- Привет, мир
- Редактирование ресурсов
- Создание меню и столбцов меню
- Создание диалогов
- Создание окон

Более внимательно вопросы использования PhAB мы рассмотрим в последующих главах.

Первые два урока охватывают основное: создание виджетов, изменение вида и поведение виджетов, генерирование кода, запуск Вашего приложения на исполнение и прочая.

Остальные уроки идут дальше базовых, показывая Вам, как создавать работающие меню, диалоги и окна. Когда Вы завершите изучение этих уроков, вы будете готовы начать проектирование почти любого приложения Photon'a.

Перед тем как Вам начать...

Вы можете запустить PhAB из меню "Launch" в левом нижнем углу экрана; выбрать подменю "Development" и затем выбрать "Builder". Вы можете также запустить его из окна консоли pterm, набрав: `appBuilder`. Для получения информации об опциях командной строки см. тему "appBuilder" в "Справочнике утилит QNX".

Перед тем как начать уроки, ознакомьтесь с интерфейсом пользователя PhAB:

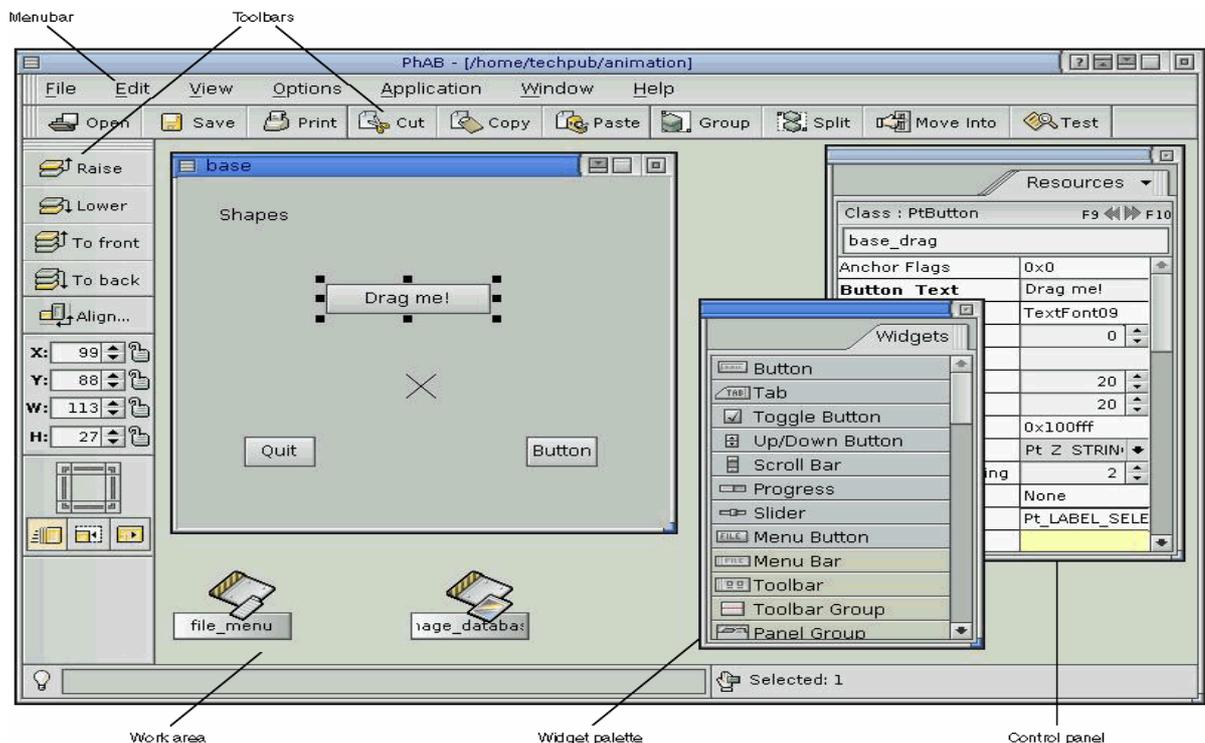


Рис. 1-1. Общий вид пользовательского интерфейса

Столбец меню	Импорт графики создание окон и диалогов, генерирование C и/или C++ кода для обеспечения всего Вашего пользовательского интерфейса в целом, и пр.
Столбцы инструментов	Экономьте время – используйте столбцы инструментов: парой щелчков мышью Вы можете копировать, перемещать, выравнивать, группировать или изменять размеры любого числа виджетов.
Рабочая область:	предоставляет Вам плоскую область, где Вы можете работать одновременно с несколькими модулями приложения.
Палитра виджетов:	позволяет легко добавлять виджеты в Ваше приложение. Просто щёлкните на нужном Вам виджете, а затем щёлкните там, где Вы хотите его разместить.
Панели управления:	позволяют Вам полностью настраивать виджеты Вашего приложения. Вы можете выбрать текстовые шрифты, модифицировать цвета, подстраивать побитовые отображения и прикреплять ответные реакции, которые будут раскрывать диалоги или активизировать приготовленный Вами C и/или C++ код.

Палитра виджетов и панели управления иницируются в одном окне, но Вы можете оттащить любое из них в отдельное окно. Для переключения между панелями в окне щёлкните на ярлыке вверху и выберите панель из меню.

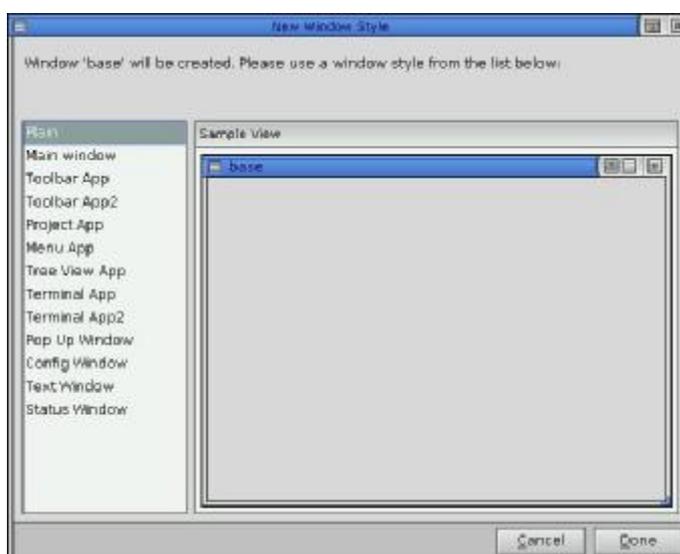
Если Вы выбрали панель управления, Вы можете переотобразить её, выбрав соответствующий пункт из меню "View".

Урок 1 – Привет, мир

На этом уроке мы научимся, как использовать PhAB для создания и компиляции простого приложения.

Создание приложения

1. Выберите в меню "File" пункт "New" для создания нового приложения. PhAB отобразит диалог, позволяющий Вам выбрать стиль принимаемого по умолчанию базового окна нового приложения:
2. Выберите стиль и щёлкните "Continue"; PhAB создаст базовое окно и отобразит его.



3. Как только Вы создали новое приложение, хорошей идеей будет сохранить его и присвоить имя. Выберите пункт "Save As" в меню "File", чтобы открыть диалог "Application Selector". Щёлкните на область "Application Name", наберите tut1, затем нажмите клавишу <Enter> либо щёлкните на "Save Application".
4. Взгляните на брусок заголовка PhAB'a. На нём теперь указывается, что имя текущего приложения – tut1.
5. Если палитра виджета не отображается, щёлкните на ярлычке вверху текущей панели управления и выберите из появившегося меню "Widgets".
6. Оторвите палитру виджетов от других панелей управления, нажав кнопку мыши на ярлыке и, удерживая кнопку нажатой, перемещайте палитру на рабочую область PhAB'a.
7. Если хотите, измените размеры палитры виджетов и панелей управления.
8. Перейдите к палитре виджетов и щёлкните на иконке виджета PtLabel:



9. Переместите указатель мыши на базовое окно приложения (указатель превратится в крестик) и щёлкните в любом месте где-нибудь возле центра окна.

PhAB автоматически выполнит следующее:

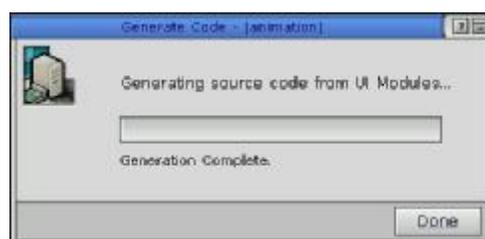
- создаст новый виджет PtLabel
 - выделит виджет, так что Вы сможете редактировать его ресурсы
 - разместит метки-манипуляторы для изменения размеров вокруг виджета
 - отобразит ресурсы виджета на контрольных панелях Resources (ресурсы) и Callbacks (ответные реакции).
10. Перейдите к панели управления Resources и подведите текст "Label1" возле ресурса "Label Text".
 11. Измените текст на "Hello World". Как только Вы это наберёте, текст на виджете изменится:



Генерирование кода

Теперь Вы готовы генерировать, компилировать и выполнять приложение.

1. Из меню "Application " выберите пункт "Build+Run". PhAB автоматически сохранит Ваше приложение и затем отобразит диалог "Build+Run".
2. Щёлкните на кнопке "Generate". Вы увидите диалог для выбора платформы – комбинацию операционной системы, компьютера и компилятора, которую Вы будете использовать. Выберите соответствующую. Например, если Вы используете ОС QNX, машину Intel x86 и компилятор gcc, выберите ptox86 и gcc.
3. Щёлкните на кнопке "Generate" для запуска процесса генерации кода. Вы увидите диалог хода работы:



Подождите, когда генерация кода завершится, затем щёлкните на кнопке "Done" для закрытия диалога.

4. Щёлкните на кнопке "Make" для компиляции кода. Вы увидите диалог "Make Application", который отобразит компилируемые файлы.
5. После того, как приложение будет откомпилировано и слинковано, станет доступна кнопка "Done" диалога. Щёлкните на ней, чтобы закрыть диалог.
6. Щёлкните на кнопке "Run", чтобы запустить Ваше новое приложение на выполнение. Приложение появится в своём собственном окне, с текстом "Hello World" в центре и принятым по умолчанию заголовком "My Application" на бруске заголовка.



Великолепно! Вы только что создали своё первое приложение в Photon'e, используя PhAB.

7. Для завершения приложения, щёлкните на кнопке меню окна в левом верхнем углу, затем выберите пункт "Close".
8. Щёлкните на кнопке "Done", чтобы закрыть диалог "Build+Run".

Хотите больше информации?

Чтобы узнать больше о компиляции, запуске и отладке приложения, см. главу "Генерирование, компилование и запуск кода на выполнение".

Урок 2. Редактирование ресурсов

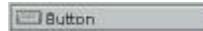
Этот урок введёт Вас в редактор ресурсов PhAB, который позволит Вам изменять вид и поведение виджетов. Вы узнаете, как редактировать фактически любую разновидность ресурса, который может иметь виджет, включая:

- количественные ресурсы (напр., ширину рамки)
- текстовые шрифты
- текстовые строки
- флаги
- цвета
- попиксельные карты (pixmaps)

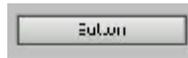
Вы также обучитесь, как создать шаблон (template), так что Вы сможете создавать другие экземпляры существующего виджета.

Добавление виджета "Кнопка"

1. Из меню "File" выберите пункт "New" для создания первого приложения. Выберите в качестве стиля "Plain window". Сохраните Ваше приложение как tut2.
2. Щёлкните на QPushButton на палитре виджета.



3. Щёлкните в центре окна приложения. Вы увидите виджет кнопки.
4. Перетащите любую метку-манипулятор изменения размера кнопки, так чтобы кнопка выглядела как на следующем рисунке:



Изменение ширины фаски

Давайте теперь изменим количественный ресурс – ширину фаски кнопки.

1. Щёлкните на ресурсе "Bevel Width" на панели управления. Вы увидите цифровой редактор:



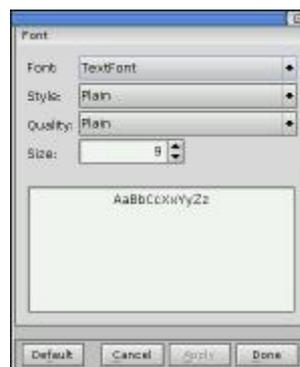
Этот редактор позволяет Вам изменить значение любого числового ресурса виджета.

2. Измените значения на 7. Чтобы сделать это, Вы можете:
 - Набрать новое значение
или
 - Щёлкать на кнопках увеличения/уменьшения
3. Для того, чтобы применить новое значение и закрыть редактор, нажмите <Enter> или щёлкните на кнопке "Done".
- Вы можете также редактировать этот ресурс (и большинство ресурсов) прямо в панели управления ресурсами. Выбирайте тот метод, который Вам нравится.

Изменение шрифта

Давайте изменим шрифт в тексте кнопки:

1. Щёлкните на ресурсе "Font". Вы увидите редактор шрифтов, который отображает текущий шрифт кнопки:



Этот редактор даёт Вам возможность изменить текстовый шрифт любого виджета, имеющего текст.

- Щёлкните на блоке "Font" или "Size", выберите тип шрифта или его размер из списка, затем щёлкните на кнопке "Apply". Текст на кнопке отобразится новым шрифтом.
- Щёлкните на кнопке "Default". Редактор отобразит шрифт виджета, принимаемый по умолчанию, но не применит шрифт к виджету.
- Если Вы хотите оставить новый выбранный Вами шрифт, щёлкните на кнопке "Cancel", чтобы проигнорировать умолчание. Если Вы хотите применить умолчание, щёлкните на кнопке "Done". В любом случае редактор шрифтов закроется.

Изменение выравнивания текста

Теперь давайте изменим горизонтальное выравнивание текста кнопки.

- Прокрутите панель управления ресурсами, чтобы найти ресурс "Horz Alignment", затем щёлкните на нём. Вы увидите редактор флагов/опций, который отображает текущее выравнивание текста виджета:



Этот редактор служит в PhAB'e двум целям:

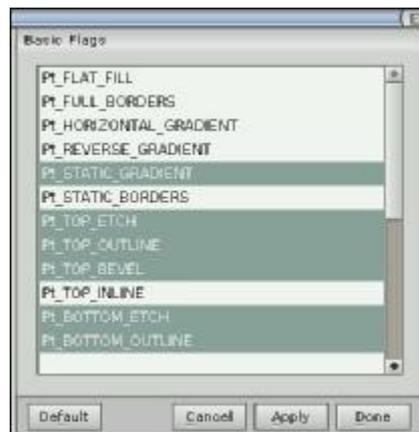
- для модификации любого ресурса – такого как выравнивание текста – который может иметь одно из нескольких предопределённых значений;
 - для выбора одного или более флагов в любом ресурсе флага
- Щёлкните на Pt_LEFT или Pt_RIGHT, затем щёлкните на кнопке "Apply". Вы увидите, что текст кнопки переместится к левому или правому краю кнопки.
 - Щёлкните на "Done".

Вы можете также установить этот ресурс непосредственно в панели управления ресурсами.

Установка флагов

Давайте теперь используем редактор флагов/опций для установки одного из флагов виджета:

- Прокрутите панель управления ресурсами, чтобы найти ресурс "Basic Flags", затем щёлкните на нём. Редактор флагов/опций переоткроется, но на этот раз он покажет текущие установки флага PtBasic:



Биты в ресурсе флага не являются взаимоисключающими, так что на это раз Вы можете, если пожелаете, использовать редактор для выбора множества опций.

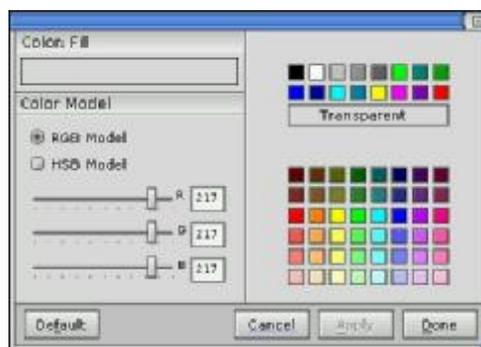
- Установите флаги `Pt_TOP_INLINE`, `Pt_BOTTOM_INLINE`, `Pt_LEFT_INLINE` и `Pt_RIGHT_INLINE`, затем щёлкните на кнопке "Done". PhAB нарисует кнопку с внутренней границей:



Изменение цвета заполнения

Давайте изменим ресурс цвета – цвет заполнения кнопки.

- Щёлкните на ресурсе кнопки "Color:Fill". Вы увидите редактор цвета, который отображает текущий цвет заполнения:



Этот редактор позволяет редактировать любой ресурс цвета. Он предлагает несколько предустановленных базовых цветов, которые будут хорошо работать со всеми графическими драйверами, и 48 настраиваемых цветов для драйверов, поддерживающих 256 и более цветов.

- Щёлкните на любом цвете из набора базовых цветов, затем щёлкните на "Apply". Кнопка окрасится в выбранный Вами цвет.



- Выберите цвет из набора настраиваемых цветов. Ползунки будут показывать значение цветов красный/зелёный/синий (RGB). Измените эти значения для получения желаемого цвета, затем примените эти изменения.

Если Вы хотите попрактиковаться с цветовой моделью цвет/насыщенность/яркость (HSB), щёлкните на кнопку модели HSB.

- Щёлкните на кнопку "Done", когда закончите экспериментировать с редактором. Ваша кнопка теперь будет окрашена в выбранный Вами цвет. Не удаляйте этот виджет; мы используем его в дальнейшем как шаблон, так что Вы сможете создавать другие такие же виджеты.

Редактирование пиксельной карты

Давайте теперь используем редактор попиксельной карты для редактирования виджета `PtLabel`. Этот редактор называется редактором попиксельной карты ("pixmap"), а не побитовой карты ("bitmap"), поскольку позволяет редактировать, кроме побитовых карт, много других типов ресурсов изображения.

Виджет `PtLabel` отображает текст и/или изображения.

1. Щёлкните на PtLabel на палитре виджетов:



2. Переместите указатель на основное окно и щёлкните под созданным Вами виджетом кнопки. Вы увидите виджет PtLabel.
3. Щёлкните на ресурсе "Label Type" в панели управления ресурсами и установите его в Pt_IMAGE.
4. Щёлкните на ресурсе "LabelImage" в панели управления ресурсами, чтобы вызвать попиксельный редактор.
5. Далее вызовите редактор цвета, чтобы выбрать цвет рисования. Просто щёлкните на следующей кнопке:

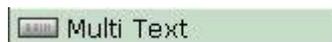


6. Выберите цвет из палитры пиксельной карты. Вы увидите, что в пиксельном редакторе немедленно изменится цвет рисования. Если Вы щёлкните на "Edit Color", Вы увидите редактор цвета, описанный ранее. Кнопки редактора цвета "Apply", "Default" и "Cancel" будут тусклыми – они не требуются для пиксельного редактора.
7. Чтобы нарисовать простое изображение, Вы можете:
 - щёлкнуть левой кнопкой мыши, чтобы заполнить ячейку цветом рисования
 - щёлкнуть правой кнопкой мыши, чтобы заполнить ячейку цветом фона
 - удерживать нажатой кнопку мыши и водить указателем, рисуя от руки.
 Можете свободно использовать другие инструменты рисования.
8. Когда Вы сделаете, что хотели, щёлкните на кнопке "Done" пиксельного редактора, чтобы принять Ваши изменения и закрыть редактор.

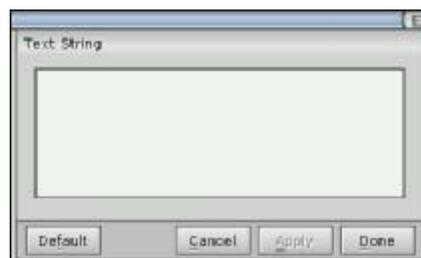
Редактирование многострочного текста

Далее мы отредактируем ресурс многострочного текста – текст виджета PtMultiText.

1. Щёлкните на PtMultiText в палитре виджетов.



2. Переместите указатель под виджет метки (label widget), который Вы только что создали, и протяните его с нажатой кнопкой мыши, так чтобы появившийся виджет PtMultiText был достаточно большим, чтобы вместить несколько строк текста.
3. Щёлкните на ресурсе "Text String" в панели управления ресурсами, чтобы вызвать редактор многострочного текста.



4. Наберите несколько строк текста. Для создания новой строки нажмите клавишу <Enter>. Например:

- Мэри имеет <Enter>
 некоего <Enter>
 маленького ягнёнка. <Enter>
- Щёлкните на кнопке "Done". Ваш текст появится в точности так, как Вы его набрали. Если это не так, попытайтесь изменить размеры виджета – виджет может оказаться недостаточной ширины или высоты.
 - Чтобы получить другой эффект, перейдите к ресурсу "Horz Aligment", щёлкните по стрелке и измените выравнивание текста на Pt_CENTER. Как Вы можете видеть, теперь каждая строка отцентрирована отдельно.
 - Если Вы ещё не наигрались, измените размеры виджета перетаскиванием одной из его меток-манипуляторов изменения размеров. Вы увидите, что текст автоматически обновится, подравниваясь под новый размер. Например:

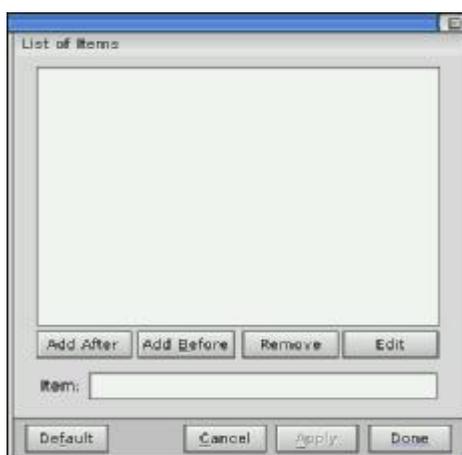


- ☞ Вы можете редактировать текст непосредственно в панели управления, но она отображает только текущую строку текста.

Редактирование списка текстовых параграфов

Давайте теперь создадим виджет PtList и добавим в виджет текст, пользуясь редактором списка. Этот редактор позволяет Вам добавлять и редактировать текст для любого виджета, который обеспечивает список текстовых параграфов.

- Щёлкните на PtList в палитре виджетов:  List
- Переместите указатель в базовое окно приложения и протащите указатель, пока новый виджет PtList не станет достаточно большим, чтобы вместить несколько строк текста.
- Щёлкните на ресурсе "List of Items", чтобы вызвать редактор списка.



- Щёлкните мышкой на текстовом блоке внизу редактора. Вы увидите курсор ввода текста.
- Наберите какой-нибудь текст, затем щёлкните на кнопку "Add After", чтобы разместить первый параграф списка.
- Теперь давайте создадим новый параграф. Щёлкните на текстовом блоке и наберите Ctrl-U, чтобы удалить текст в текстовом блоке, затем наберите какой-нибудь новый текст.

- Щёлкните на "Add After", чтобы разместить этот новый параграф после предыдущего.
7. Повторяйте предыдущий шаг, пока Вам не надоест.
 8. Щёлкните на "Apply". Виджет PtList будет теперь отображать созданный Вами список.
 9. Теперь попытаемся отредактировать список:
 - чтобы изменить существующий параграф, щёлкните на параграфе, отредактируйте его текст, затем щёлкните на "Edit";
 - чтобы удалить параграф, щёлкните на нём, а затем щёлкните на "Remove".
 10. Когда Вы закончите свои эксперименты, щёлкните на "Done", чтобы принять Ваши изменения и закрыть редактор.

Создание шаблона

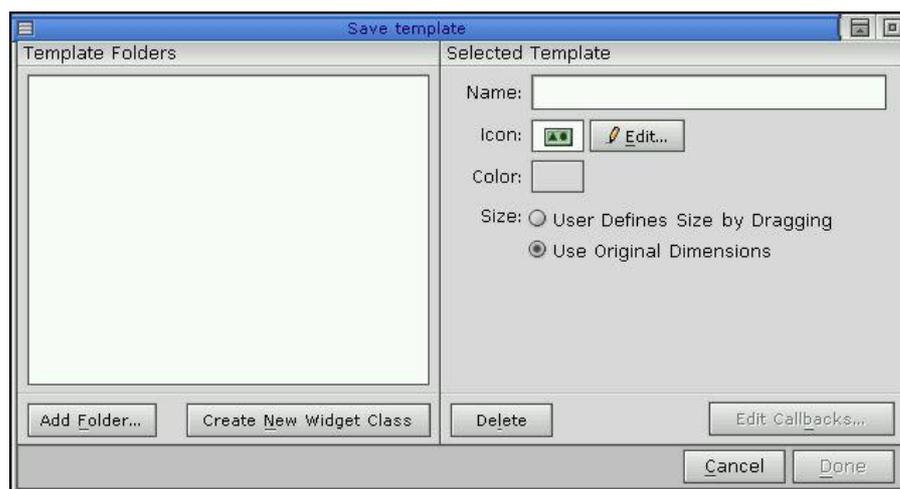
Иногда Вам может понадобиться создать много виджетов, которые бы выглядели и вели себя сходным образом. Вы можете делать это путём создания виджета, редактированием его ресурсов с последующим копированием его, однако это не всегда достаточно удобно.

PhAB упрощает это, позволяя Вам создавать шаблон из существующего виджета или виджетов. PhAB создаёт палитру, схожую с палитрой виджетов, для Ваших шаблонов.

Давайте создадим шаблон из кнопки, которую Вы создали ранее на этом уроке.

1. Начните с выбора кнопки.
2. Щёлкните на меню "Edit", и затем выберите "Save as template".

Появится следующий диалог:

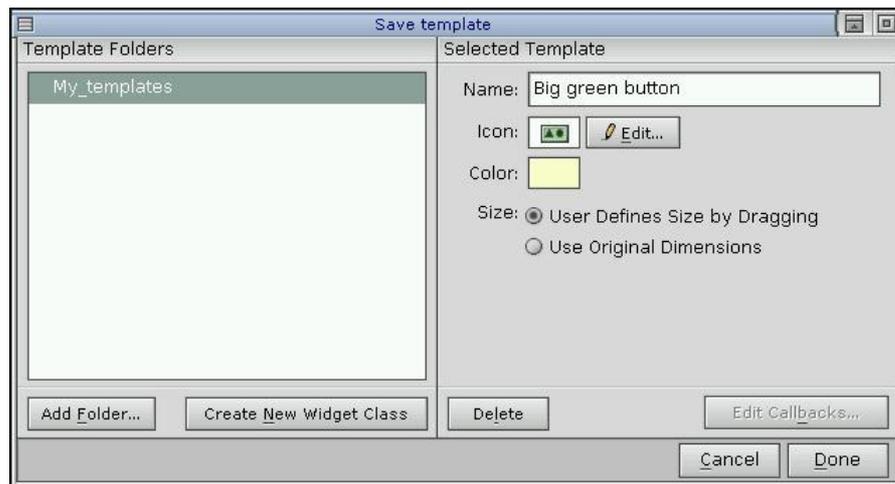


3. Вам необходимо создать папку, в которой Вы будете хранить шаблоны, поэтому щёлкните на "Add Folder...". Отобразится диалог:



4. Наберите My_templates как имя папки, затем щёлкните на "Add". Диалог закроется, и имя папки отобразится в диалог "Save template".
5. Дайте имя шаблону, такое как "Big green button". Это имя, которое PhAB будет использовать в палитре.

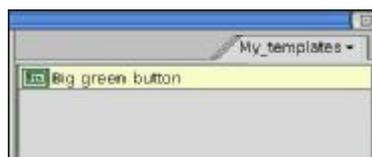
6. Можете при желании создать иконку для шаблона на входе палитры. Щёлкните на "Edit Icon" и затем следуйте инструкциям, данным ранее для редактирования попиксельных карт. Вы можете сделать иконку, выглядящей как виджет.
7. Можете при желании выбрать цвет фона для входа в палитре, щёлкнув в блоке "Color". Вы можете использовать различные фоновые цвета в палитре, чтобы различать виджеты, предназначенные для различных целей (напр., кнопки и текстовые виджеты).
8. Выберите метод изменения размеров. Это определяет, будете ли Вы протаскивать мышью или прямо щёлкать при создании экземпляров Вашего шаблона. Для этой кнопки выберите метод протягивания ("User Define Size by Dragging" на подписи к кнопке).
9. Диалог теперь будет выглядеть таким образом:



Щёлкните на "Done".

Вы только что создали шаблон! Теперь давайте посмотрим, как его использовать.

- Щёлкните на меню "View" и затем выберите "Palettes". Появится каскадное меню Ваших палитр. Заметьте, что оно включает "My_templates", и что Ваш шаблон будет автоматически выбран.
- Щёлкните где-нибудь вне меню, чтобы закрыть его.
- Перейдите к панелям управления и щёлкните на верхнем ярлыке. Всплывающее меню теперь включает "My_templates"; выберите его, чтобы отобразить палитру.



- Щёлкните на иконке Вашей настраиваемой кнопки, создайте её экземпляр и отредактируйте, как показано:



Если пожелаете, можете сохранить, сгенерировать, собрать и запустить на выполнение приложение.

Всегда, когда Вы запустите PhAB, он автоматически загрузит палитру "My_templates". Вы можете использовать вход "Palettes" в меню "View", чтобы отменить это.

Желаете узнать побольше?

Вам теперь известны базовые сведения по редактированию любых ресурсов виджета в PhAB. Для получения более полной информации см. следующие разделы в главе "Редактирование ресурсов и ответных реакций".

ЧТОБЫ РЕДАКТИРОВАТЬ:	СМ. РАЗДЕЛ:
Побитовые карты и изображения	Попиксельный редактор
Цвета	Редактор цвета
Флаги	Редактор флагов/опций
Шрифты	Редактор шрифтов
Список текстовых параграфов	Редактор списков
Числа	Редактор чисел или редактор флагов/опций
Одно- и многострочные тексты	Текстовые редакторы

Для получения более полной информации по шаблонам см. раздел "Шаблоны" в главе "Создание виджетов".

Урок 3. Создание меню и панелей меню

Этот урок проведёт Вас шаг за шагом по пути создания меню и панелей меню.

О присоединении ответных реакций

На этом уроке Вы научитесь, как устанавливать связь с ответной реакцией, одним из ключевых компонентов PhAB. Для понимания того, что есть связь с ответной реакцией, давайте начнём с небольшой "фоновой" информации об ответных реакциях виджета.

Почти все виджеты поддерживают разнообразные ответные реакции. Эти ответные реакции позволяют интерфейсу Вашего приложения взаимодействовать с кодом Вашего приложения. Например, давайте скажем, что Вы хотите, чтобы Ваше приложение выполняло действие, когда пользователь щёлкает на кнопку. В этом случае Вы можете прикрепить функцию ответной реакции к ответной реакции "Activate" кнопки.

В некоторых window-образных приложениях Вы можете прикрепить к ответным реакциям виджетов только функции в виде кодов. Но когда Вы используете PhAB для создания ответной реакции, Вы можете сделать на шаг больше и прикрепить целые окна, диалоги, меню и многое другое. Это та расширенная функциональность, которую мы называем присоединением ответной реакции.

PhAB предоставляет два основных типа присоединения ответной реакции:

- Модульный тип присоединения ответной реакции
Прикрепление модуля приложения (такого, как окно, диалог или меню) к любой ответной реакции виджета. Модуль открывается каждый раз, когда встречается состояние ответной реакции. На этом уроке Вы присоедините модуль меню к ответной реакции кнопки "Arm".
- Кодовый тип присоединения ответной реакции.
Присоединение функции в виде кода к любой ответной реакции виджета. Виджет вызывает функцию всякий раз, когда встречается состояние ответной реакции. Заметьте, что некоторые присоединения ответной реакции кодового типа позволяют Вам автоматически закрыть родительский модуль. На этом уроке Вы присоедините кодовую функцию к ответной реакции пунктов меню.

Об именах экземпляров

Для доступа к виджету из кода Вашего приложения Вы должны прежде всего присвоить виджету имя экземпляра. Поскольку все имена экземпляров виджета находятся в одном глобальном пространстве имён, внутри приложения не может быть двух виджетов, имеющих одно и то же имя экземпляра.

Мы рекомендуем Вам начинать каждое имя экземпляра с префикса модуля. Например, Ваше базовое окно имеет виджет `PtButton`, который содержит текст "Blue" в качестве метки, Вы можете дать этому виджету имя экземпляра `base_blue` (а лично я – так и вообще бы `Base_Button_Blue`. Прим. пер., любящего длинные названия).

☞ Принятие соглашения по именам для Ваших виджетов сделает проще Вашу работу с большими приложениями.

Создание панели меню

Чтобы научиться привязывать ответные реакции, давайте создадим два функционирующих меню – "File" и "Help" – которые Вы сможете потом включить в ваши собственные приложения.

В PhAB'e меню делаются из двух кусков:

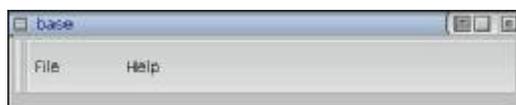
1. кнопки меню, которой Вы щёлкните, чтобы отобразить меню;
2. модуля меню, который содержит пункты меню.

Используя привязывание ответных реакций, Вы свяжете модули меню с кнопками "File" и "Help" на панели меню. Вы также свяжете ответную реакцию кодового типа с пунктом меню "Quit" в модуле меню "File". Эта ответная реакция позволяет пункту "Quit" закрывать приложение.

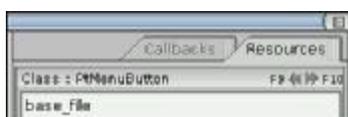
1. Из меню "File" PhAB выберите пункт "New" для запуска нового приложения. Выберите стиль "Plain Window".
2. Сохраните приложение как `tut3` (По информации о сохранении Вашего приложения см. предыдущий урок или раздел "Сохранение приложения" в главе "Работа с приложениями").
3. Выберите из палитры виджетов виджет `PtMenuBar`, укажите на верхний левый угол панели основного окна, и протащите мышку, чтобы панель меню стала в ширину окна. Панель меню увеличивается и уменьшается при изменении ширины окна и всегда располагается вверху окна. Вы можете видеть это, щёлкнув на панели заголовка окна и изменяя его размеры при перетаскивании одной из его меток-манипуляторов изменения размеров.

☞ Если Вы случайно щёлкнули на кнопке "Test" на правом конце панели заголовка модуля, окно не будет изменять размеры или принимать новые виджеты. Если это случилось, просто щёлкните ещё раз на кнопку "Test".

После того, как Вы завершите следующее действие, панель меню будет выглядеть таким образом:



4. Разместите виджет `PtMenButton` на только что созданной Вами панели меню. Кнопка меню автоматически центрируется по вертикали на панели меню.
5. Перейдите на панель управления ресурсами и щёлкните на имени экземпляра виджета сразу после имени класса. Измените имя экземпляра кнопки на `base_file`:



6. Измените ресурс "Label Text" кнопки на "File".

7. Разместите другой виджет PtMenuButton следом за первым. Измените имя экземпляра на base_help и текст на "Help".

Создание модуля меню "File"

Теперь, когда Вы имеете кнопки меню, Вам необходимо создать модули меню. Давайте начнём с меню "File".

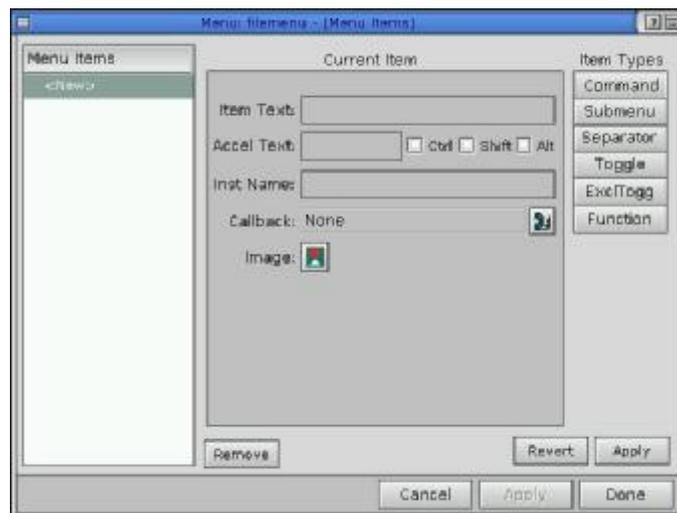
1. Из меню "Application" выберите "Menus", чтобы открыть переключатель модулей. Этот переключатель позволит Вам создать или просмотреть любой тип модуля PhAB.
2. В блоке "Name" наберите filemenu, нажмите <Enter>. Поскольку модуля меню ещё не существует, PhAB спросит Вас, создавать ли модуль. Щёлкните на "Yes".
Вы увидите, что модуль меню появится в Вашей рабочей области и имя модуля в прокручиваемом списке переключателя модулей.
3. Переключатель модулей остаётся на экране, позволяя вам создавать модули дальше. Однако, Вам надо создать пока только одно меню, поэтому щёлкните "Done", чтобы закрыть переключатель.

Добавление пунктов меню

Давайте теперь добавим несколько пунктов меню в меню "File".

- ☞ Если Вы щёлкните на другом модуле, модуль меню станет невыбранным, что означает, что Вы не можете с ним работать. Чтобы вновь выбрать модуль меню, щёлкните на его панели заголовка.

1. Щёлкните на ресурсе "Menu Items" на панели управления ресурсами. Вы увидите редактор меню:



Если Вы посмотрите на список пунктов меню, то увидите, что выбран пункт "New". Этот специальный пункт позволяет Вам добавлять пункты в меню.

2. Чтобы добавить Ваш первый пункт меню – который, так уж случилось, тоже будет называться "New" – щёлкните на область "Item Text", затем наберите "New".
3. Теперь присвойте пункту имя экземпляра. В области "Inst Name" наберите file_new.
4. Щёлкните на "Apply", чтобы добавить пункт в меню. Вы увидите имя пункта в списке "Menu Items", предварённое префиксом CMD. Префикс CMD означает, что это командный пункт; то есть пункт, вызывающий ответную реакцию PhAB'a.
5. Повторите вышеприведенные шаги для создания пунктов меню, помеченных "Save" и "Save As". Присвойте этим пунктам имена экземпляров file_save и file_as.
6. До сих пор Вы добавляли пункты меню командного типа. Теперь добавьте пункт типа разделитель. Просто щёлкните на кнопке "Separator" возле верхнего правого угла. Вы увидите список стилей разделителя:

separator Style	
<input type="radio"/> Single Line	⇒ Одиночная линия
<input type="radio"/> Double Line	⇒ Двойная линия
<input type="radio"/> Single Dash Line	⇒ Одиночная прерывистая линия
<input type="radio"/> Double Dash Line	⇒ Двойная прерывистая линия
<input type="radio"/> Etched - In	⇒ Вдавленный
<input type="radio"/> Etched - Out	⇒ Выпуклый
<input type="radio"/> Blank	⇒ Пустой

Что означает:

- ⇒ Одиночная линия
- ⇒ Двойная линия
- ⇒ Одиночная прерывистая линия
- ⇒ Двойная прерывистая линия
- ⇒ Вдавленный
- ⇒ Выпуклый
- ⇒ Пустой

7. Выберите стиль или просто щёлкните на "Apply", чтобы получить принимаемый по умолчанию стиль, каковой является "Etched - In".
8. Теперь давайте добавим пункт "Quit". Щёлкните на кнопке "Command", затем зададим "Quit" в тексте пункта и file_quit как имя экземпляра.
9. Вы завершили работу с модулем меню, так что щёлкните "Done". Модуль отобразит созданные Вами пункты:



10. Вам понадобится аккуратно попридерживать этот модуль в сторонке, пока Вы будете работать над своей следующей задачей. Поэтому щёлкните на кнопке минимизации приложения (левая кнопка на правой стороне панели заголовка) или выберите кнопку меню "Work" (верхний левый угол) и выберите "Minimize".

Создание модуля меню "Help"

Используйте то, чему Вы научились при создании модуля меню, выполнив следующее:

1. Создайте Ваш модуль меню "Help" и дайте ему имя helpmenu.
 2. В этом модуле разместите один командный пункт, названный "About Demo" и присвойте пункту имя экземпляра help_about. Когда закончите, минимизируйте модуль.
- ☞ Если один из модулей Вашего меню кажется "пропавшим" (Вы можете нечаянно закрыть его или поместить его позади другого модуля), его легко вновь сделать видимыми. См. раздел "Нахождение пропавших модулей и иконок" в главе "Работа с модулями".

Присоединение ответных реакций

Давайте вернёмся к кнопкам меню, которые Вы создали раньше, и прикрепим ответные реакции, так что кнопки смогут вызывать всплытие Ваших модулей меню.

Присоединение ответной реакции модульного типа

- Выберите кнопку меню "File", затем переключитесь на панель управления ответными реакциями ("Callbacks"). Вы увидите список ответных реакций кнопки "File".



Чтобы модуль меню "File" всплывал при нажатии кнопки "File", Вам надо присоединить ответную реакцию "Arm" к кнопке. Для прикрепления ответной реакции "Arm" Вы можете открыть меню, используя механизмы "щёлкнуть-переместить-щёлкнуть" либо "нажать-перетащить-опустить". Щёлкните на "Arm", чтобы вызвать редактор ответной реакции.

Область "Module Types" редактора позволяет Вам выбрать тип модуля, к которому Вы хотите присоединиться. Поскольку Вы хотите присоединить кнопку "File" к модулю меню, щёлкните на "Menu".

Щёлкните на области "Name" и наберите filemenu, что является именем, которое Вы дали Вашему модулю меню "File". Это подсоединит кнопку меню к этому модулю. Вы можете также выбрать filemenu из всплывающего списка доступных модулей. Чтобы вызвать список, щёлкните на иконке справа от области "Name".

Щёлкните на "Apply", чтобы добавить связь с ответной реакцией, затем щёлкните на "Done", чтобы закрыть редактор ответных связей.

Повторите вышеприведенные шаги для присоединения кнопки меню "Help" к модулю меню "Help".

Присоединение ответной реакции кодового типа

Давайте теперь присоединим ответную реакцию кодового типа к пункту "Quit" меню "File", так чтобы она могла завершать работу приложения.

1. Выполните двойной щелчок на свёрнутом в иконку модуле filemenu. Он откроется и выберет модуль.
2. Переключитесь на панель управления ресурсами и щёлкните на ресурсе "Menu Items".
3. Выберите пункт "Quit" в списке "Menu Items".
4. Щёлкните на иконке следом за областью "Callback", чтобы открыть редактор ответной реакции:



5. Когда редактор откроется, принятым по умолчанию типом ответной реакции будет "Code". Поскольку это нужный Вам тип, всё, что Вам надо сделать – это задать имя функции, которую Вы хотите вызвать. Функция должна иметь имя, имеющее смысл (святые слова! Сколько уже вызывал я функций с именами типа fl или pro... Прим. пер.) Так что наберите quit в области "Function".
6. Щёлкните на "Apply", чтобы обновить список ответных реакций, затем щёлкните на "Done", закрывая редактор.
7. Щёлкните на "Done" вновь, чтобы закрыть редактор меню.

Подготовка кода

Вы теперь генерируете код для Вашего приложения и отредактируете сгенерированную заглушку кода, так чтобы пункт "Quit" приводил к завершению приложения.

- Из меню "Application" выберите "Build+Run" и сгенерируйте код приложения.
- После того, как процесс генерации завершился, диалог "Build+Run" отобразит список сгенерированных файлов.
Прокрутите список в поисках файла quit.c. Это сгенерированный шаблон кода, который PhAB сгенерировал для Вашей функции quit ().
- Вам необходимо сделать функцию выхода из программы. Чтобы сделать это, выберите quit.c из списка файлов, щёлкните на кнопке "Edit" и измените функцию quit (), как показано ниже:

```
int
quit( PtWidget_t *widget, ApInfo_t *apinfo,
      PtCallbackInfo_t *cbinfo )
{
    /* предотвращает предупреждения (warnings) об отсутствии ссылок */
    widget = widget,
    apinfo = apinfo,
    cbinfo = cbinfo;

    PtExit( EXIT_SUCCESS );

    /* Этот оператор не может быть достигнут, */
    /* но это оставит компилятор счастливым */

    return( Pt_CONTINUE );
}
```

PtExit () – это функция, которая очищает окружение Photon'a и затем закрывает приложение. Она описана в "Справочнике библиотеки Photon'a".

- После того, как Вы отредактировали код, сохраните Ваши изменения, закройте редактор и щёлкните на "Make" для компиляции кода.
- Щёлкните на "Run Application" для запуска приложения.
- После запуска приложения щёлкните на кнопку "File" для вызова меню "File". Затем выберите "Quit". Ваше приложение немедленно завершит работу и закроется.

Хотите узнать больше?

ЧТОБЫ УЗНАТЬ БОЛЬШЕ:	СМ. РАЗДЕЛ:	В ГЛАВЕ:
Ответные реакции виджета	Ответные реакции	Редактирование ресурсов и ответных реакций в PhAB
Имена экземпляров	Имена экземпляров	Создание виджетов в PhAB
Модули меню	Модули меню	Работа с модулями

Урок 4. Создание диалогов

Этот урок описывает, как создать диалог. Он также предоставит хороший пример того, как Вы можете использовать установочный код для модификации ресурсов виджета перед тем, как виджет появится на экране.

☞ Этот урок использует приложение, созданное Вами на уроке 3.

На это уроке Вы:

1. присоедините пункт "About Demo" в меню "Help" к диалогу
2. добавьте надписи и кнопку "Done" к новому диалогу
3. определите установочную функцию, которая изменит текст одной из надписей для отображения номера версии, когда диалог будет вызван.

О диалогах

Модули диалогов спроектированы для того, чтобы позволить Вам получить дополнительную информацию о пользователе. Обычно Вы используете такую информацию, чтобы выполнить конкретную команду или задачу.

Поскольку Вам обычно нет нужды получать одну и ту же информацию дважды, диалоги являются модулями, существующими в одном экземпляре. Иными словами, Вы не можете открыть один и тот же диалог более одного раза одновременно. Если Вы попытаетесь создать второй экземпляр диалога, PhAB просто переведёт существующий диалог поверх остальных окон и передаст ему фокус.

Если Вам необходимо создать окно, поддерживающее несколько экземпляров, используйте модуль окна. Вы изучите модули окна на следующем уроке.

Ещё об именах экземпляров

Чтобы облегчить доступ к виджетам из кода Вашего приложения, PhAB генерирует глобальную переменную и декларацию. Обе они базируются на имени экземпляра виджета.

Глобально переменная, начинающаяся с префикса `ABN_`, представляет собой имя виджета. Декларация, начинающаяся с префикса `ABW_`, представляет указатель на экземпляр виджета. Например, пусть Вы имеете виджет, имя которого `about_version`. PhAB использует это имя для генерации глобальной переменной с именем `ABN_about_version` и декларации с именем `ABW_about_version`.

На этом уроке мы научимся, как использовать эти сгенерированные имена.

☞ Значение переменной виджета `ABN_...`, является уникальным для всего приложения.

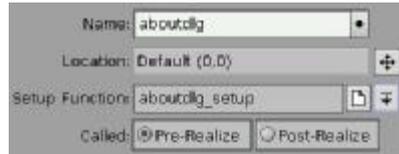
Прикрепление модуля диалога

1. Откройте приложение `tut3`, созданное Вами, и используйте пункт "Save As" в меню "File", чтобы сохранить его как `tut4`.
2. Откройте созданный Вами модуль меню "Help" (он может быть ещё свёрнутым в иконку).
3. Щёлкните на ресурсе "Menu Items" в панели управления ресурсами, чтобы открыть редактор меню.
4. Выберите пункт "About Demo", затем щёлкните на иконке, следующей за областью "Callback", чтобы открыть редактор ответных реакций:



5. Когда редактор откроется, принимаемым по умолчанию типом ответной реакции будет "Code". Перейдите к группе "Module Types" и измените тип ответной реакции на "Dialog".

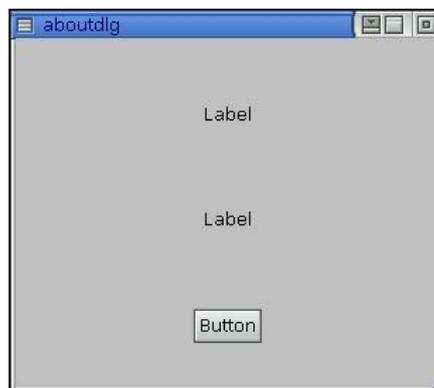
6. В области "Name" наберите как имя модуля диалога, с которым Вы хотите установить связь, aboutdlg. (Этот диалог ещё не существует, но позже PhAB спросит Вас, создавать ли его).
7. В области "Setup Function" наберите about_setup. Это имя, которое мы присваиваем функции установки, которая будет вызвана перед запуском диалога.
Используя эту функцию, мы изменим содержание виджета надписи внутри диалога для отображения номера версии.
8. Поскольку Вы хотите, чтобы функция aboutdlg_setup вызывалась перед открытием диалога, убедитесь, что включена кнопка "Pre-Realize".
9. Щёлкните на иконке "Location", чтобы задать, где должен появляться диалог при его запуске. (Хорошим выбором будет расположение "Center Screen"). Щёлкните на "Done". Информация об ответной реакции будет теперь выглядеть так:



10. Щёлкните на "Apply", чтобы добавить связь ответной реакции. Поскольку модуль диалога, к которому Вы хотите прикрепить связь, ещё не существует, Phab предложит Вам выбрать стиль; выберите "Plain" и щёлкните "Continue". Вы увидите в рабочей области новый диалог. Вы также увидите новую ответную реакцию в списке ответных реакций в редакторе ответных реакций.
11. Щёлкните на "Done", чтобы закрыть редактор ответных реакций, затем щёлкните на "Done" ещё раз, чтобы закрыть редактор меню.

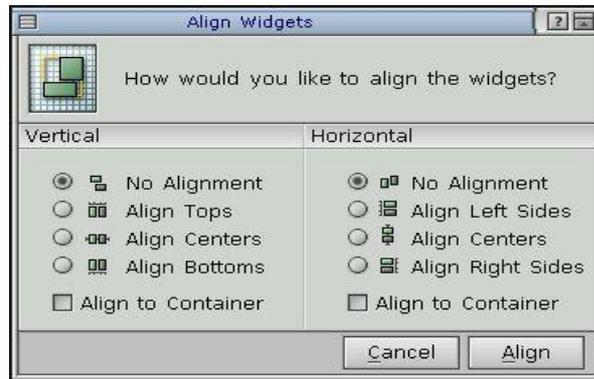
Добавление виджетов в диалог

1. Откройте модуль диалога aboutdlg
2. Поместите два виджета PtLabel в верхней половине диалога, и виджет PtButton в нижней части:

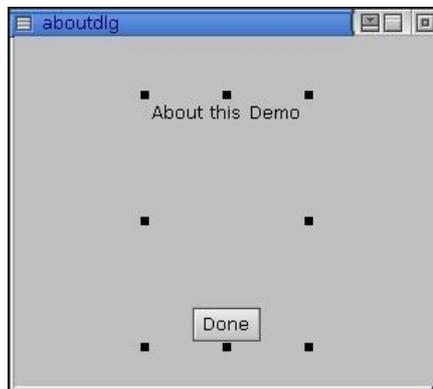


3. Выберите верхний виджет PtLabel и измените ресурс его текста на "About this Demo". Затем измените его горизонтальное выравнивание на Pt_CENTER.
4. Выберите второй виджет PtLabel и измените его текст на пустую строку. Затем измените его горизонтальное выравнивание на Pt_CENTER.
Позже Вы заполните его с помощью функции aboutdlg_setup(), так что она изменит пустой текст этой надписи на отображение номера версии.
5. Вы должны присвоить этому пустому виджету PtLabel имя экземпляра, поскольку будете ссылаться на него в коде программы. Поэтому измените его имя экземпляра на about_version.
6. Выберите виджет PtButton и измените ресурс текста кнопки на "Done". Затем измените его имя экземпляра на about_done.

7. Давайте отцентрируем виджеты горизонтально в диалоге. Выберите оба виджета PtLabel и виджет PtButton и выберите из меню "Edit" пункт "Alignment". Вы увидите диалог "Align Widgets".



8. В столбце "Horizontal" щёлкните на "Align Centers" и на "Align to Container". Затем щёлкните на кнопке "Align". Две надписи и кнопка теперь будут горизонтально отцентрированы внутри Вашего диалога. Ваш модуль about должен теперь выглядеть так:



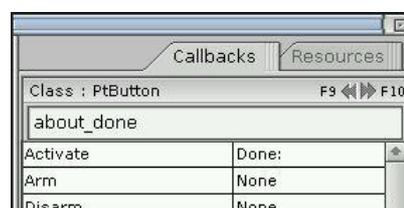
Добавление ответной реакции к кнопке Done

Теперь добавим ответную реакцию к кнопке Done, так чтобы диалог закрывался, когда пользователь щёлкал на этой кнопке.

1. Выберите кнопку "Done", затем переключитесь на панель управления ответными реакциями (панель "Callbacks").
2. Щёлкните на "Activate", чтобы добавить ответную реакцию на активизацию. Вы увидите редактор ответной реакции.
3. Выберите тип "Done code", затем щёлкните на "Apply". Ничего не вводите в области "Function".

Выбор типа "Done code" говорит виджету исполнять операцию "Done" при активизации виджета. То есть виджет вызывает функцию, заданную в области "Function" (если она задана) и затем закрывает модуль диалога.

4. Закройте редактор. Список ответных реакций теперь указывает, что Вы добавили ответную реакцию активизации ("Activate"), называемую Done:



Модификация сгенерированного кода функции

Теперь Вы измените сгенерированный код функции `about_setup()`, так чтобы она изменяла текст надписи `about_version`, показывая номер версии.

1. Сохраните Ваше приложение.
2. Откройте диалог "Build+Run" и сгенерируйте код.
3. Когда генерация кода завершится, закройте диалог "Generate Code", выберите файл `aboutdlg_setup.c` из списка файлов и щёлкните на кнопке "Edit"² (или дважды щёлкните на имени файла).

Измените код с

```
int aboutdlg_setup( PtWidget_t *link_instance,
                  ApInfo_t *apinfo,
                  PtCallbackInfo_t *cbinfo )
{
    /* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    link_instance = link_instance,
        apinfo = apinfo,
        cbinfo = cbinfo;

    return( Pt_CONTINUE );
}
```

на следующий:

```
int aboutdlg_setup( PtWidget_t *link_instance,
                  ApInfo_t *apinfo,
                  PtCallbackInfo_t *cbinfo ) {
    /* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    link_instance = link_instance, apinfo = apinfo, cbinfo = cbinfo;

    PtSetResource( ABW_about_version, Pt_ARG_TEXT_STRING,
                  "1.00", 0);

    return( Pt_CONTINUE );
}
```

Код поместит номер версии (1.00) в ресурс текстовой строки виджета `about_version`. Чтобы сделать это, код вызывает `PtSetResource()`, чтобы установить значение ресурса виджета `about_version`. Код использует сгенерированную PhAB'ом декларацию `ABW_about_version`, которая обеспечивает доступ к указателю на экземпляр виджета. Мы можем безопасно использовать эту декларацию, поскольку имеем дело с модулем диалога – PhAB гарантирует, что в данное время будет существовать только один экземпляр диалога.

4. Сохраните Ваши изменения и закройте текстовый редактор.

Компиляция и запуск на выполнение

Теперь Вы готовы компилировать и запускать программу.

1. Щёлкните кнопку "Make". Если Ваша программа откомпилирована и слинкована без ошибок (что и будет, если Вы корректно редактировали функцию), щёлкните на кнопку "Run" для запуска приложения.
2. Из запущенного приложения откройте меню "Help" и выберите пункт "About Demo". Откроется диалог и Вы увидите номер версии (1.00) под надписью "About this Demo". Заметьте, что диалог появится в заданном Вами месте.
3. Теперь попытайтесь вызвать второй экземпляр диалога. Как Вы видите, это не работает. PhAB всегда гарантирует, что существует только один экземпляр виджета диалога.

² В среде разработки редактором по умолчанию является `red`. Редактор не очень удобен и – нет подсветки и тому подобных прелестей. В этом смысле значительно приятнее использовать редактор `Workspace` (<http://pages.infnit.net/micbel/>). Сменить редактор по умолчанию можно в настройках (`preferences`). Подробнее это описано в Главе 2: Подгонка вашего окружения PhAB.

4. Щёлкните на "Done", чтобы закрыть диалог, затем завершите приложение, выбрав пункт "Quit" из меню "File". Наконец, закройте диалог "Build+Run".

Желаете узнать больше?

ЧТОБЫ УЗНАТЬ БОЛЬШЕ О:	СМ. РАЗДЕЛ:	В ГЛАВЕ:
Использовании диалога	Модули диалога	Работа с модулями
Именах экземпляров	Имена экземпляров Переменные и декларации	Создание виджетов в PhAB Работа с кодом
Ответных реакциях	Ответные реакции Код функций ответных реакций	Редактирование ресурсов и ответных реакций в PhAB Работа с кодом
Генерирование кода	Генерирование кода приложения	Генерирование, компиляция и запуск кода на выполнение

Урок 5. Создание окон

На предыдущем уроке Вы научились, как оперировать модулями диалога, которые поддерживают только один экземпляр. На этом уроке Вы изучите, как оперировать модулями окна, которые поддерживают множественность экземпляров.

☞ На этом уроке используется приложение, созданное Вами на уроке 4.

Для поддержания множественности экземпляров модули окна обеспечивают большую гибкость, нежели диалоги. Но чтобы получить преимущества этой гибкости, Вы должны поддерживать связь с каждым указателем на экземпляр окна. Это гарантирует Вам, что Вы корректно ссылаетесь на виджеты внутри каждого экземпляра окна. Вы не можете безопасно использовать сгенерированную глобальную декларацию `ABW_xxx`, поскольку она ссылается только на последний созданный экземпляр.

Для упрощения задачи работы с множественными экземплярами PhAB обеспечивает функции библиотеки API, позволяющие Вам получить доступ к любому виджету через его глобальную переменную – имя (`ABN_xxx`).

Создание окна

Для начала давайте создадим модуль окна и прикрепим его к пункту "New" меню "File" в tut4. Это окно будет содержать кнопки, которые изменяют цвет другого виджета.

На предыдущем уроке Вы создали модуль диалога из редактора ответных реакций. Но на этот раз Вы используете переключатель модулей (`module selector`), чтобы создать требуемый Вам модуль. В дальнейшем используйте тот метод, который Вам больше нравится.

1. Откройте приложение tut4, если Вы удалили его из рабочей области PhAB.
2. Сохраните приложение как tut5.
3. Сверните в иконку модуль диалога `aboutdlg`.
4. В меню "Application" выберите пункт "Windows", чтобы открыть переключатель модулей.
5. В области "Name" наберите `newwin` для имени экземпляра окна, затем нажмите `<Enter>` или щёлкните на "Open". Когда PhAB предложит Вам выбрать стиль окна, выберите "Plain" и щёлкните "Continue".
6. Закройте переключатель модулей. Модуль окна будет теперь выбранным элементом.

Прикрепление ответной реакции

Поскольку модуль окна поддерживает множественность экземпляров, Вы создаёте код функции, которая будет вызвана всякий раз, когда окно будет открыто или закрыто (т.е. всякий раз, когда окно создаётся или уничтожается). Так что давайте прежде всего установим ответную реакцию, определяющую, когда окно закрывается:

1. Переключитесь, если необходимо, на панель управления ответными реакциями.
2. Из списка ответных реакций выберите "Window Manager". Вы хотите использовать ответную реакцию менеджера окон, поскольку она вызывается, когда Photon'овский менеджер окон закрывает окно.
3. В зоне "Function" наберите `newwin_close`. Вы не выбираете тип обратной реакции, поскольку принимаемый по умолчанию "Code" – это то, что Вы хотите. Щёлкните на "Apply", затем на "Done".
4. Переключитесь на панель управления ресурсами и выберите ресурс "Flags: Notify". Убедитесь, что флаг `Ph_WM_CLOSE` установлен (т.е. подсвечен), затем щёлкните на "Done". Этот флаг указывает менеджеру окон уведомлять Ваше приложение, когда окно закрывается.
5. Теперь давайте установим функцию, которая будет вызываться при открытии окна. Откройте модуль меню `filemenu`, затем выберите ресурс "Menu Items" в панели управления ресурсами. Вы увидите редактор меню.
6. Убедитесь, что в списке пунктов меню "Menu Items" выбран пункт "New", затем щёлкните на иконке ответной реакции, чтобы открыть редактор ответных связей.
7. Выберите тип модуля "Window", затем щёлкните на стрелке возле области "Name". Вы увидите список существующих модулей окон.
8. Выберите `newwin`, который является окном, только что Вами созданным.
9. В области "Setup Function" введите `newwin_setup` как имя установочной функции. В дальнейшем Вы модифицируете `newwin_setup`, чтобы манипулировать множественными экземплярами окон.
10. Щёлкните "Apply", затем "Done". Щёлкните ещё раз "Done", чтобы закрыть редактор меню.

Добавление виджетов

Давайте теперь добавим несколько виджетов в модуль окна `newwin`. Используя эти виджеты, Вы научитесь, как обновлять информацию в текущем или другом экземпляре модуля окна.

- Добавьте виджет `PtRect` и четыре виджета `PtButton`, как показано ниже:



- Теперь модифицируем левую кнопку:
 - Изменим текст надписи кнопки на `Red`.
 - Присвоим кнопке имя экземпляра `btn_red`
 - Прикрепим ответную реакцию "Activate", зададим тип кода "Code" и имя функции `color_change`.
- Модифицируем среднюю кнопку:

- Изменим текст надписи кнопки на Green.
 - Зададим имя экземпляра кнопки btn_green.
 - Прикрепим Activate/Code ответную реакцию к той же функции, что и выше – color_change.
 - Модифицируем правую кнопку:
 - Изменим текст надписи кнопки на Blue.
 - Зададим имя экземпляра кнопки btn_blue.
 - Прикрепим ответную реакцию типа Activate/Code к той же функции, что и выше – color_change.
 - Модифицируем большую кнопку:
 - Изменим текст надписи на "Change previous window's color".
 - Зададим имя экземпляра кнопки btn_prev.
 - Прикрепим ответную реакцию типа Activate/Code к той же функции, что и выше color_change.
 - Наконец, присвоим прямоугольнику имя экземпляра color_rect. Вам необходимо задать это имя, так чтобы функция color_change() могла изменить цвет прямоугольника.
- Ваше окно теперь будет выглядеть таким образом:



Генерирование и модификация кода

На последнем уроке Вы использовали сгенерированную декларацию ABW_xxx, чтобы получить доступ к указателю на экземпляр диалога. При работе с множественными экземплярами модуля окна Вы не можете использовать эту декларацию, поскольку она ссылается только на последнее созданное окно. Вместо этого Вы добавляете код к сгенерированной функции установки окна, так чтобы она хранила копию каждого указателя на экземпляр окна в глобальном массиве виджетов. На этом уроке Вам понадобятся эти указатели для работы кнопки "Change Previous Window Color".

Генерирование кода

Откройте диалог "Build+Run" и регенерируйте код.

Модификация функции установки

Теперь давайте модифицируем функцию newwin_setup(), так чтобы она :

- ограничивала число возможных экземпляров пятью;
- хранила копии каждого указателя на окно;
- отображала номер экземпляра окна на панели заголовка этого окна.

Отредактируйте файл newwin_setup.c, как показано ниже:

```
int          win_ctr = 0;
PtWidget_t *win[5];

int
newwin_setup( PtWidget_t *link_instance,
```

```

        ApInfo_t *apinfo,
        PtCallbackInfo_t *cbinfo )

{
    char    buffer[40];

    /* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    link_instance = link_instance, apinfo = apinfo, cbinfo = cbinfo;

    /* Заметьте: Возвращение Pt_END в предреализационной функции установки
       указывает PhAV удалять модуль без его реализации */

    /* позволяет только 5 окон максимум */
    if ( win_ctr == 5 ) {
        return( Pt_END );
    }

    /* сохранение указателя на экземпляр модуля окна */
    win[win_ctr] = link_instance;

    sprintf( buffer, "Window %d", win_ctr + 1 );
    PtSetResource( win[win_ctr], Pt_ARG_WINDOW_TITLE,
                  buffer, 0 );

    win_ctr++;

    return( Pt_CONTINUE );
}

```

Модификация функции изменения цвета

Теперь давайте модифицируем функцию `color_change()`, так чтобы :

- нажатие кнопки "Red", "Green" или "Blue" изменяло цвет прямоугольника в цвет кнопки;
- нажатие на кнопку "Change Previous Window Color" изменяло фон предыдущего окна на цвет из массива.

☞ Если бы это был модуль диалога, Вы бы могли использовать декларацию `ABW_color_rect`, чтобы обновить цвет прямоугольника. Однако, поскольку это модуль окон, Вы должны использовать указатель на экземпляр окна, в котором нажата кнопка.

Чтобы получить указатель на экземпляр виджета текущего окна, Вам необходимо вызвать:

1. `ApGetInstance()` для получения указателя на окно, содержащее виджет, вызвавший ответную реакцию
2. `ApGetWidgetPtr()` для получения указателя на виджет с данной декларацией `ABN_...`

Если гарантировано существование только одного экземпляра окна, нижеследующее будет работать:

```
PtSetResource (ABW_color_rect, Pt_APG_FILL_COLOR, buffer, 0);
```

Но в рассматриваемом случае `color_change` должно использовать:

```
PtSetResource (ApGetWidgetPtr (ApGetInstance (winget), ABN_color_rect),
              Pt_APG_FILL_COLOR, buffer, 0);
```

Поэтому Вам необходимо изменить `color_change.c` таким образом:

```

PgColor_t    colors[5] = {Pg_BLACK, Pg_YELLOW, Pg_MAGENTA,
                          Pg_CYAN, Pg_DGREEN};

int          base_clr = -1;
extern int   win_ctr;
extern PtWidget_t *win[5];

int
color_change( PtWidget_t *widget, ApInfo_t *apinfo,
              PtCallbackInfo_t *cbinfo )
{
    int    i, prev;
    PtWidget_t *this_window;

    /* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    widget = widget, apinfo = apinfo, cbinfo = cbinfo;

    /* Получает указатель на текущее окно */

```

```

this_window = ApGetInstance( widget );

if ( ApName( widget ) == ABN_btn_red ) {
    PtSetResource(
        ApGetWidgetPtr( this_window, ABN_color_rect ),
        Pt_ARG_FILL_COLOR, Pg_RED, 0 );
} else if ( ApName( widget ) == ABN_btn_green ) {
    PtSetResource(
        ApGetWidgetPtr( this_window, ABN_color_rect ),
        Pt_ARG_FILL_COLOR, Pg_GREEN, 0 );
} else if ( ApName( widget ) == ABN_btn_blue ) {
    PtSetResource(
        ApGetWidgetPtr( this_window, ABN_color_rect ),
        Pt_ARG_FILL_COLOR, Pg_BLUE, 0 );
} else if ( ApName( widget ) == ABN_btn_prev ) {

/* Заметьте: Здесь мы используем указатели на экземпляры модулей окон, сохранённые в
newwin_setup, чтобы обновить предыдущее окно до текущего в случае, когда оно не закрыто.
Определяется, какое окно является предыдущим по отношению к этому окну */

    prev = -1;
    for ( i = 0; i < win_ctr; i++ ) {
        if ( win[i] == this_window ) {
            prev = i - 1;
            break;
        }
    }

/* Если окно по прежнему существует, обновляет его цвет фона. */

    if ( prev != -1 && win[prev] ) {
        base_clr++;
        if (base_clr >= 5) {
            base_clr = 0;
        }
        PtSetResource( win[prev], Pt_ARG_FILL_COLOR,
            colors[base_clr], 0 );
    }
}

return( Pt_CONTINUE );
}

```

Модификация функции закрытия окна

И в завершение Вам необходимо модифицировать функцию `newwin_close()`, так чтобы она устанавливала массив `win` указателей на экземпляры в `NULL` для окна, когда оно закрывается. Таким способом Вы можете проверять на `NULL` в массиве `win`, чтобы определить, существует ли ещё окно.

Модифицируйте код `newwin_close.c`, как показано выше:

```

extern int      win_ctr;
extern PtWidget_t *win[5];

int
newwin_close( PtWidget_t *widget, ApInfo_t *apinfo,
              PtCallbackInfo_t *cbinfo )
{
    PhWindowEvent_t *we = cbinfo->cbdata;
    int i;

/* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    apinfo = apinfo;

/* обработка только события WM close */
    if ( we->event_f != Ph_WM_CLOSE ) {
        return( Pt_CONTINUE );
    }
}

```

```

/* Прелестно, это закрыто. Тогда что это такое? */
for ( i = 0; i < win_ctr; i++ ) {
    if ( win[i] == widget ) {
        win[i] = NULL;
        break;
    }
}
return( Pt_CONTINUE );
}

```

Компиляция и запуск

- Соберите приложение и запустите его.
- В меню "File" приложения выберите несколько раз пункт "new", чтобы создать несколько окон. Вы увидите на панели заголовка окна соответствующий номер этого окна.
- Щёлкните на кнопке цвета, чтобы изменить цвет прямоугольника. Затем щёлкните на кнопке "Change Previous Window Color" на любом окне, чтобы изменить цвет фона предыдущего окна.

Хотите узнать больше?

ЧТОБЫ УЗНАТЬ БОЛЬШЕ О:	СМ. РАЗДЕЛ	В ГЛАВЕ:
Использование окон	Модули окон	Работа с модулями
Имена экземпляров	Имена экземпляров Переменные и декларации	Создание виджетов в PhAB Работа с кодом
Ответные реакции	Ответные реакции Коды функций ответных реакций	Редактирование ресурсов и ответных реакций в PhAB Работа с кодом
Генерация кода	Генерация кода приложения	Генерация, компиляция и запуск кода на выполнение
События окна	Флаги управления окном	Управление окном

Глава 2. Окружение PhAB

В этой главе более детально описано окружение PhAB и то, как Вы можете его настраивать.

Глава включает разделы:

- Меню
- Панели инструментов
- Панели управления
- Палитра виджетов
- Панель ресурсов
- Панель ответных реакций
- Панель дерева модуля
- Панель связей модуля
- Панель поиска
- Настройка Вашего окружения PhAB

Меню

Вверху рабочей области PhAB Вы можете видеть следующую панель меню:

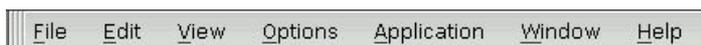


Рис. 2-1. Панель меню PhAB

Меню "File"

Команды, имеющие дело с Вашими приложениями и его файлами:

"New"	Создание нового приложения, см. раздел "Создание приложения" в главе "Работа с приложениями".
"Open"	Открытие существующего приложения, см. "Открытие приложения" в главе "Работа с приложениями". Эта команда доступна также через панели инструментов PhAB'a
"Save"	
"Save As"	Сохранение текущего приложения под тем же или другим именем; см. "Сохранение приложения" в главе "Работа с приложениями". Команда "Save" доступна также через панели инструментов PhAB'a.
"Close"	Закреть текущее приложение, см. "Заккрытие приложения" в главе "Работа с приложениями".
"Import Files"	Импортировать файлы, созданные в других приложениях, см. "Импортирование модулей PhAB из других приложений" в главе "Работа с приложениями".
"Exit"	Завершить Вашу текущую сессию в PhAB. PhAB запросит Вас, если имеются какие-либо несохранённые Вами изменения.

В этом меню также приводится список последних нескольких приложений, которые Вы редактировали.

Меню "Edit"

Команды по редактируемым виджетам:

"Cut", "Copy",	Удалить и скопировать виджеты в буфер обмена, и "вклеить" их оттуда, см.
"Paste"	"Буфер обмена" в главе "Создание виджетов в PhAB".
"Transfer"	Переместить виджет с одного контейнера в другой, см. "Перемещение виджетов между контейнерами" в главе "Создание виджетов в PhAB".

"Delete"	Удаляет виджет без сохранения в буфере обмена, см. "Удаление виджетов" в главе "Создание виджетов в PhAB".
"To Front", "To Back"	Перемещает выбранные виджеты на первый или на задний план в контейнере, см. "Выстраивание виджетов" в главе "Создание виджетов в PhAB".
"Group Together", "Split Apart"	Комбинирует выбранные виджеты в группу или распускает выбранную группу, см. "Выравнивание виджетов с использованием группирования" в главе "Управление геометрией"
"Aligment"	Выравнивает выбранные виджеты; см. "Выравнивание виджетов" в главе "Создание виджетов в PhAB".
"Change Class"	Изменяет класс выбранных виджетов, см. "Изменение класса виджета" в главе "Создание виджетов в PhAB".
"Save as template", "Edit templates"	Шаблон – это созданный пользователем виджет, который он хочет использовать как основу для других виджетов. Эти команды позволяют Вам создать или редактировать шаблон; см. "Шаблоны" в главе "Создание виджетов в PhAB".

Многие из этих команд также имеются в инструментальных панелях PhAB.

Меню "View"

Команды, отображающие необязательные окна PhAB:

"Clipboard"	Местоназначение виджетов, когда они вырезаются или копируются; см. "Буфер обмена" в главе "Создание виджетов PhAB".
"Resourses", "Callbacks", "Module Tree", "Module Links", "Search Panel"	Панели управления для редактируемых ресурсов, ответных реакций и прочая, см. "Панели управления".
"Palettes"	Каскадные меню палитр, включая палитру виджета и любых созданных шаблонов, см. "Панели управления"

Меню "Options"

Команды, отображающие диалоги для задаваемых опций:

"Preferences"	Предпочтение PhAB, такие как цвета, команды редактирования и стили имён ресурсов.
"Grid"	Необязательная сетка, которую можно использовать для позиционирования виджетов.
"Generate report"	Генерирует отчёт о виджетах и модулях приложения.

Для получения более полной информации см. "Подгонка Вашего окружения PhAB."

Меню "Application"

Команды, имеющие дело с приложениями в целом:

"Startup Info/Modules"	Информация, используемая для приложения в целом, включая глобальные хедеры (заголовочные файлы), функцию инициализации и то, какие модули отображать при запуске. Для получения более полной информации см. "Задание информации запуска приложения" в главе "Работа с приложениями".
"Languages"	Меню команд, используемых для создания многоязычных версий Вашего приложения; см. главу "Поддержка международных языков"
"Windows", "Dialogs", "Menus", "Pictures", "Icons"	Диалоги, приводящие список модулей Вашего приложения и позволяющие Вам создавать новые модули. Для получения более полной информации см. главу "Работа с модулями".
"Internal Links"	Внутренняя связь – это механизм PhAB, позволяющий Вам получить доступ к модулю PhAB непосредственно из кода Вашего приложения; см. главу "Получение доступа к модулям PhAB из кода".
"Build+Run"	"Центр управления" компиляции и запуска Вашего приложения на исполнение.
"Generate"	Генерация кода для Вашего приложения.

"Convert to Multiplatform"

Эта команда ("Конвертирование в многоплатформенность") полезна только для старых приложений, которые были сгенерированы под версию 4 ОС QNX для однозначной платформы. Для получения более полной информации см. главу "Генерирование, компилирование и запуск программы на исполнение".

Меню "Window"

Команды, которые манипулируют окнами PhAB:

"Arrange Modules"

Размещает модули Вашего приложения так, что они укладываются от верхнего левого к нижнему правому углу рабочего пространства PhAB.

"Arrange Icons"

Размещает свёрнутые в иконки модули в рабочем пространстве PhAB в алфавитном порядке построено.

Это меню также предоставляет список всех модулей Вашего приложения. Выбор одного из этих пунктов меню разворачивает модуль, если тот был свёрнут в иконку, и выбирает модуль.

Меню "Help"

Получает онлайн-справочную информацию "Welcome to PhAB", "Tutorials", "PhAB Concepts", "Tools+Techniques". Связывает с соответствующей секцией данного руководства программиста.

"PhAB Library API"

Связь со "Справочником библиотеки Photon".

"About PhAB"

Номер версии и копирайтная информация о PhAB.

В PhAB доступны и другие формы получения помощи:

- контекстная помощь – чтобы получить помощь в части пользовательского интерфейса PhAB, щёлкните на кнопке со знаком вопроса, затем на интересующем Вас пункте. Хэлпвьювер отобразит информацию о выбранном пункте;
- всплывающая помощь – чтобы узнать, для чего нужна та или иная кнопка на палитре виджетов или панели инструментов, задержите на ней указатель мыши. Появится всплывающее описание.

Панели инструментов

Панели инструментов дают Вам возможность быстрого доступа к часто используемым командам из панели меню:

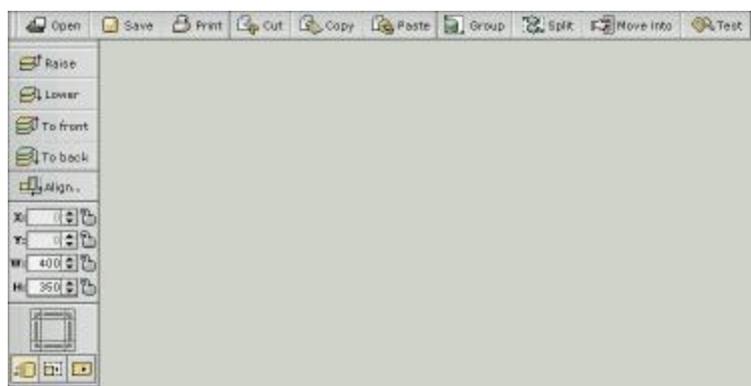


Рис. 2-2. Панели инструментов PhAB

"Open"	Открыть существующее приложение; см. "Открытие приложения" в главе "Работа с приложениями". Эта команда также доступна из меню "File".
"Save"	Сохранить текущее приложение, см. "Сохранение приложения" в главе "Работа с приложениями". Команда также доступна из меню "File".
"Print"	Не обеспечена
"Cut", "Copy", "Paste"	Удалить и скопировать виджеты в буфер обмена, и вклеить затем из него; см. "Буфер обмена" в главе "Создание виджетов в PhAB". Эти команды также доступны из меню "Edit".
"Group", "Split"	Объединить выбранные виджеты в группу или разбить выбранную группу, см. "Выравнивание виджетов с использованием группирования" в главе "Управление геометрией". Эти команды также доступны через меню "Edit".
"Move Into"	Переместить виджет из одного контейнера в другой; см. "Перемещение виджетов между контейнерами" в главе "Создание виджетов в PhAB". Эта команда соответствует команде "Transfer" из меню "Edit".
"Test"	Переключает в режим тестирования, так что Вы можете взаимодействовать с виджетом таким образом, как будто Ваше приложение запущено на выполнение.
"Raise", "Lower", "To front", "To back"	Переместить выбранные виджеты на передний или задний план; или вперёд, или назад в контейнере, см. "Выстраивание виджетов" в главе "Создание виджетов в PhAB". Команды "To front" и "To back" также доступны через меню "Edit".
"Align"	Наиболее часто используемые команды выравнивания выбранных виджетов; см. "Выравнивание виджетов" в главе "Создание виджетов в PhAB". Для более полного выбора возможностей выравнивания см. пункт "Alignment" в меню "Edit".
"X", "Y", "W", "H"	Координаты и размер выбранного в текущий момент виджета. Чтобы изменить их, наберите новое значение и нажмите <Enter>. Чтобы сделать невозможным изменение координат или размеров текущего виджета, закройте их, щёлкнув на изображении замочка, так чтобы он закрылся. Вы не сможете изменить область (ни вводом значения, ни перетаскиванием), пока не отожрёт его. Замочки сохраняются вместе с Вашими приложениям.

Инструмент корректировки положения

Этот инструмент позволяет Вам переместить, растянуть или ужать виджет. Щёлкните на кнопке желаемого режима и затем щёлкайте на рамочных кнопках, расположенных выше.

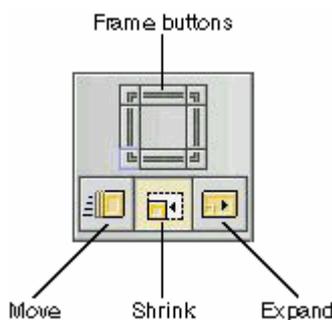


Рис. 2-3. Компоненты инструмента корректировки положения

Каждый щелчок на рамочных кнопках сдвинет, растянет или сожмёт выбранный виджет на один пиксель. Для сдвижки на несколько пикселей удерживайте кнопку мыши нажатой.

☞ Вы можете также использовать клавишу <Ctrl> и цифровую (вспомогательную) клавиатуру для пододвигания, растягивания или сжатия виджета. Каждая клавиша

соответствует одной из кнопок инструмента корректировки. Нажатие <Ctrl>+<5> переключает режимы, и <Ctrl>+<↑> работает как верхняя рамочная кнопка.

Панели управления

PhAB включает набор панелей управления, отображающих информацию о текущем выбранном виджете или виджетах. Они отображаются по умолчанию в PhAB, и Вы можете перемещать их куда Вам заблагорассудится. Если Вы закроете панель управления, Вы сможете открыть её вновь, выбрав соответствующий пункт из меню "View".

Панели управления включают:

- Палитру виджетов
- Панель ресурсов
- Панель ответных реакций
- Панель дерева модулей
- Панель связей модулей
- Панель поиска

Они описаны в нижеследующих разделах.

Панели управления первоначально отображаются как пачка в виджете PtPanelGroup. Если Вы щёлкните на ярлыке панели, появится меню панелей. Если Вы достаточно расширили окно, все ярлычки отобразятся в линию.

Вы можете вытащить панели из группы, чтобы настроить рабочее место. Если Вы бросите её на фон рабочей области PhAB, она станет новой группой панелей. Если Вы перебросите её в другую группу панелей, панель присоединится к этой группе. Затем Вы вольны изменить размеры групп панелей, как Вы найдёте нужным. В зависимости от Вашего выбора в диалоге "AppBuilder Preferences Settings" ("Установка предопределений построителя приложений"), размещение панелей будет сохранено с Вашим приложением или для всех Ваших сеансов работы в PhAB.

Палитра виджетов

Палитра виджетов позволяет Вам добавлять виджеты в Ваше приложение.



Рис. 2-4. Палитра виджетов PhAB

Если Вы закроете эту панель, Вы сможете вновь открыть её, выбрав пункт "Palettes" из меню "View" и затем "Widgets" из всплывшего меню.

Виджеты выстроены и обозначены различными цветами в соответствии с типами. Имена необязательны; чтобы скрыть или отображать их, щёлкните правой клавишей на палатре и выберите соответствующий пункт во всплывшем меню.

Чтобы выяснить, какой виджет представляет кнопка, если имена виджетов не отображаются:

- Задержите указатель мыши на ней, пока не всплывёт подсказка или
- См. приложение "Обзор виджетов".

Для получения информации об использовании определённых классов виджетов см. "Справочник виджетов Photon".

Режимы (создание или выбор)

Палитра виджетов имеет два режима:

- Режим выбора Позволяет Вам выбрать существующие виджеты и модули на рабочей области.
- Режим создания Позволяет Вам создавать новые виджеты.

Определение режима

Чтобы определить, в каком Вы режиме:

- Посмотрите на палитру виджетов – если кнопка иконки вдавлена, Вы в режиме создания.
- Посмотрите на указатель – если указатель представляет из себя обычную стрелку с остриём, когда Вы перемещаете его по рабочей области, Вы в режиме выбора. Если указатель выглядит иначе, Вы в режиме создания.

Переключение в режим создания

Чтобы переключиться в режим создания, щёлкните на любой иконке виджета на палитре виджета. Теперь Вы сможете создать один или более экземпляров этого виджета. Для получения более полной информации см. раздел "Создание виджетов" в главе "Создание виджетов в PhAB".

Переключение в режим выбора

Для переключения из режима создания в режим выбора, выполните одно из следующих действий:

- Щёлкните на фоне рабочей области PhAB или
- щёлкните правой клавишей мыши на модуле или
- щёлкните на выбранном виджете на палитре виджетов.

По умолчанию PhAB возвращается в режим выбора, как только Вы создали виджет.

Панель ресурсов

Панель ресурсов отображает список ресурсов для выбранного виджета или виджетов. (Если выбрано более одного виджета, эта панель отображает только ресурсы, которые они имеют сообща). Вот пример:



Рис. 2-5. Панель ресурсов

Если Вы закрыли эту панель, Вы можете вновь открыть её, выбрав пункт "Resources" из меню "View".

Она включает следующее:

Класс виджета	Класс выбранного виджета
Кнопки "следующий" и "предыдущий"	Позволяют Вам последовательно перемещаться по виджетам в текущем модуле. Эти кнопки позволяют Вам также выбрать несколько виджетов или выбирать виджеты внутри группы. Для получения более подробной информации см. раздел "Выбор виджетов" в главе "Создание виджетов в PhAB".
Имя экземпляра	Позволяет Вам ввести уникальное имя экземпляра для виджета. Для получения более полной информации см. раздел "Имена экземпляров" в главе "Создание виджетов в PhAB".

Вы можете изменить значение ресурсов прямо в панели управления, либо Вы можете использовать редактор, предоставляющий полные возможности, щёлкнув на имени ресурса. Для получения более полной информации см. главу "Редактирование ресурсов и ответных реакций".

По умолчанию панели управления "Resources" и "Callback" отображают названия ресурсов описательно. Если Вы задержите указатель на ресурсе, на всплывающей подсказке отобразится заголовочная декларация (header manifest).

Чтобы получить надписи, отображающие текущие заголовочные декларации (что удобно при написании кода), откройте диалог "Preferences" и измените установку в области "Resource Names". Чтобы открыть этот диалог, выберите пункт "Preferences" из меню "Options". Теперь, если Вы задержите указатель мыши на ресурсе, всплывающая надпись отобразит описание.

☞ Панель управления не отображает все ресурсы виджета. PhAB автоматически устанавливает Pt_ARG_AREA, Pt_ARG_DIM, Pt_ARG_EXTENT и Pt_ARG_POS при перемещении или изменении размеров виджета. Некоторые другие ресурсы слишком сложны, чтобы редактировать их в PhAB.

Панель ответных реакций

Панель ответных реакций отображает список ресурсов ответных реакций выбранного виджета. Вы можете использовать эту панель только когда Вы выбрали один виджет. Виджет должен иметь уникальное имя экземпляра. Вот пример:

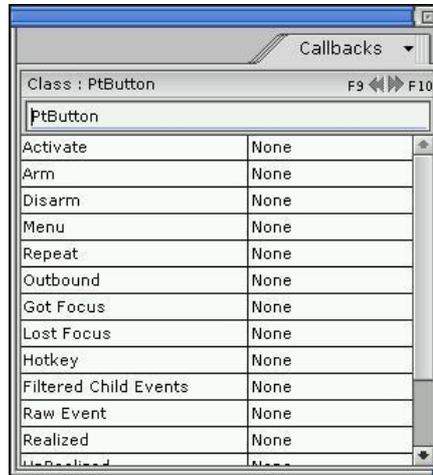


Рис. 2-6. Панель ответных реакций

Если Вы закрыли эту панель, Вы можете вновь открыть её, выбрав "Callback" из меню "View". Эта панель, подобно панели ресурсов, отображает класс виджета и имя экземпляра, и кнопки перехода на следующий и предыдущий виджеты.

На левой стороне списка указывается тип ответной реакции. На правой отображается:

- "None" если нет ответных реакций
- Тип ответной реакции и имя, если имеется одна ответная реакция
- Число ответных реакций, если их более одной.

Чтобы создать ответную реакцию или отредактировать существующую, щёлкните на соответствующем ресурсе (напр., Pt_CB_ACTIVATE).

Панель дерева модулей

Панель дерева модулей отображает иерархическое дерево виджетов в текущем модуле. Вот пример:

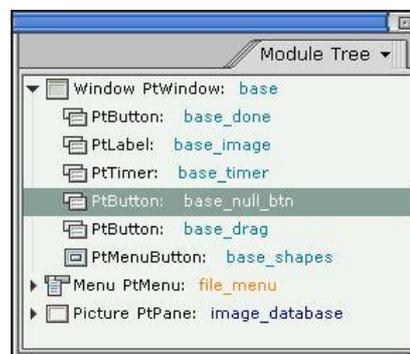


Рис. 2-7. Панель дерева модулей

Если Вы закрыли эту панель, вы можете вновь открыть её, выбрав "Module Tree" из меню "View".

Эта панель позволяет легко:

- просмотреть родительские/потомков связи виджетов модуля;
- выбрать виджет внутри группы;
- найти виджет по имени;
- выбрать виджет, скрытый под другим виджетом.

Чтобы выбрать виджет на дереве, щёлкните на имени виджета. Если Вы щёлкните на этой панели правой кнопкой мыши, появится меню:

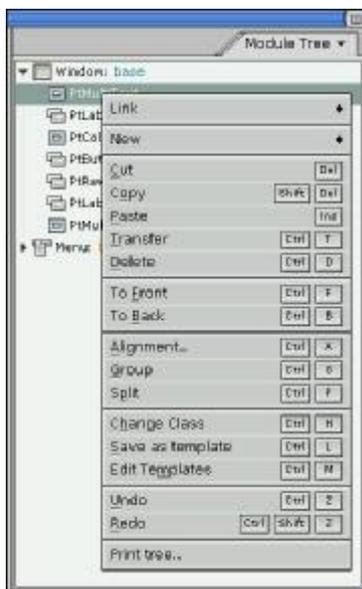


Рис. 2-8. Меню панели дерева модулей

Панель связей модуля

Панель связей модуля отображает список всех связанных ответных реакций – как *к*, так и *от* текущего модуля. Как Вы можете видеть на нижеследующем примере, ответные реакции отображаются в формате двух строк:



Рис. 2-9. Панель связей модуля

Чтобы:	Щёлкните на:
выбрать виджет	имени экземпляра (напр., <code>base_file</code>) в строке 1
редактировать ответную реакцию виджета	соответствующем типе ответной реакции (напр., <code>Arm</code>) в строке 2

Если Вы закрыли эту панель, Вы можете открыть её, выбрав пункт "Module Links" в меню "View".

Панель поиска

Панель поиска позволяет Вам найти виджет в Вашем приложении по заданному типу, имени, текстовому ресурсу и прочая.

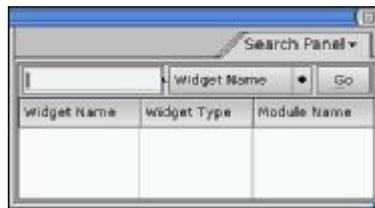


Рис. 2-10. Панель поиска

Если Вы закрыли эту панель, Вы можете вновь её открыть, выбрав "Search Panel" из меню "View". Просто выберите категорию, которую Вы хотите найти, из комбинированного окна и задайте шаблон (который является чувствительным к регистру вводимых букв):

Widget Name	В текстовой области наберите точное имя виджета или регулярное выражение. Например, значение <code>my_button*</code> совпадает со всеми виджетами, имя которых начинается с <code>my_button</code> .
Widget Type	Наберите имя класса или регулярное выражение (напр., <code>PtScroll*</code>), или используйте комбинированное окно для выбора класса виджет.
Widget Text	Наберите определённый текст или регулярное выражение для просмотра текстовых ресурсов виджетов.
Callback Type	Поиск виджетов, имеющих прикрепленные ответные реакции типа (Code, Done и прочая), выбранного из комбинированного окна шаблона.
Callback Function Name	Наберите имя функции или регулярное выражение.
Callback Module Name	Наберите имя модуля или регулярное выражение. Будут выбраны все виджеты, имеющие ответные реакции, указывающие на модуль, чьё имя совпадает с шаблоном.

Наконец, нажмите кнопку "Go". В списке отобразятся виджеты, удовлетворившие критерию поиска. Выберите вход из списка, чтобы выбрать искомые виджеты; модули PhAB, в которых те расположены, откроются или станут видимыми.

Подгонка Вашего окружения PhAB

Чтобы подогнать PhAB под свои предпочтения:

1. Выберите пункт "Preferences" из меню "Options". Вы увидите диалог установки предпочтений ("Preference Settings").



Рис. 2-11. Установка предпочтений PhAB

2. Щёлкните на кнопке, представляющей вид установок, которые Вы хотите изменить: "General", "Colors" или "Dragging".
3. Когда Вы завершите установку, щёлкните по "Done".

Общие предпочтения ("General preferences")

Вы можете установить следующие общие предпочтения:

Workspace	Позволяет Вам выбрать, сохранять ли Ваши предпочтения отдельно для каждого приложения, или для каждого пользователя, или не сохранять.
Resource Names	По умолчанию, панели ресурсов и ответных реаукций отображают надписи ресурсов описательно. Эта область позволяет Вам отображать надписи как текущие заголовочные декларации, что Вы можете найти полезным при написании кода. Заметьте, однако, что декларации длинные и занимают на экране больше места. Если Вы задержите указатель на ресурсе, надпись, не отображённая в панели управления, будет отображаться во всплывающем тексте.
Icon Descriptions	По умолчанию, когда Вы задержите указатель мыши на любой иконке, всплывёт описывающий текст. Чтобы отключить эту возможность, щёлкните на "None".
Edit Command	Позволяет Вам задать редактор для использования с диалогом "Build+Run".
View Command	Позволяет Вам задать просмотрщик файлов для использования с диалогом "Build+Run".
Print Command	Позволяет Вам задать команду печати, используемую для печати выбранного файла (напр., в диалоге "Build+Run").
Automatically save	Определяет, сохранять или не сохранять приложение автоматически, и если сохранять, то как часто.
Warnings on exiting	Предупреждать или нет Вас, когда Вы выходите из PhAB без генерирования кода или сохранения Вашего приложения.

Предпочтение цвета ("Color preferences")

Вы можете установить следующие предпочтения цвета:

Resize Handle	
Non-Resizable Handle	Если Вы выбрали цвет фона окна, который делает плохо видимыми метки-манипуляторы изменения размеров, используйте эти опции для подгонки цвета (если Вы выберете виджет и метки-манипуляторы изменения размеров появляются окрашенными в цвет, соответствующий неизменности размеров, размеры виджета нельзя изменить).

Предпочтение по перетаскиванию (Dragging preferences)

Вы можете установить следующие предпочтения по перетаскиванию:

Widget Module	Перетаскивать виджеты и модули как контуры, а не как полные объекты.
Drag Damping Factor	(Демпфирующий фактор перетаскивания). Величина того, на сколько Вы должны перетащить виджет или модуль перед тем, как тот начнёт перемещаться. Этот фактор позволяет избежать перемещения виджета, когда на самом деле Вы хотели просто его выбрать.

Предпочтения сетки

Для позиционирования и задания размеров сетки Вы можете использовать сетку. Чтобы изменить сетку, выберите пункт "Grid" из меню "Options". Появится следующий диалог:

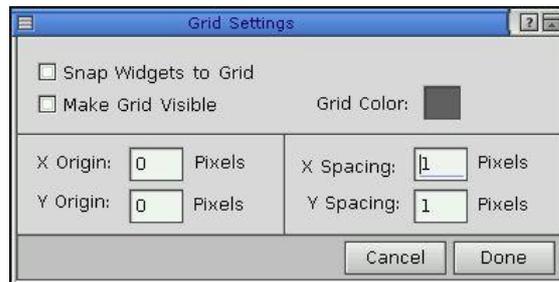


Рис. 2-12. Диалог установки регуляторов сетки

Этот диалог позволяет Вам:

- Прикрепить положение нового виджета к сетке
- Сделать сетку видимой
- Выбрать цвет сетки
- Задать начало координат и интервал `ctnrb`

☞ Установки сетки остаются только для текущей сессии. Они не сохраняются.

Глава 3. Работа с приложениями

Эта глава описывает работу в PhAB с приложениями в целом. Она включает:

- Создание приложения
- Открытие приложения
- Сохранение приложения
- Закрытие приложения
- Задание стартовой информации приложения
- Импортируемые файлы

Для получения информации по запуску приложения на выполнение см. главу "Генерирование, компиляция и запуск программы на исполнение".

Создание приложения

Чтобы создать новые приложения, выберите "New" из меню "File" или нажмите <Ctrl> + <N>. Если Вы уже работаете с приложением, PhAB спросит Вас, желаете ли Вы сохранить сделанные в этом приложении изменения, перед тем как закрыть его.

PhAB создаёт новое безымянное приложение, состоящее из простого основного окна по имени base. PhAB отображает диалог, в котором Вы можете выбрать стиль базового окна.

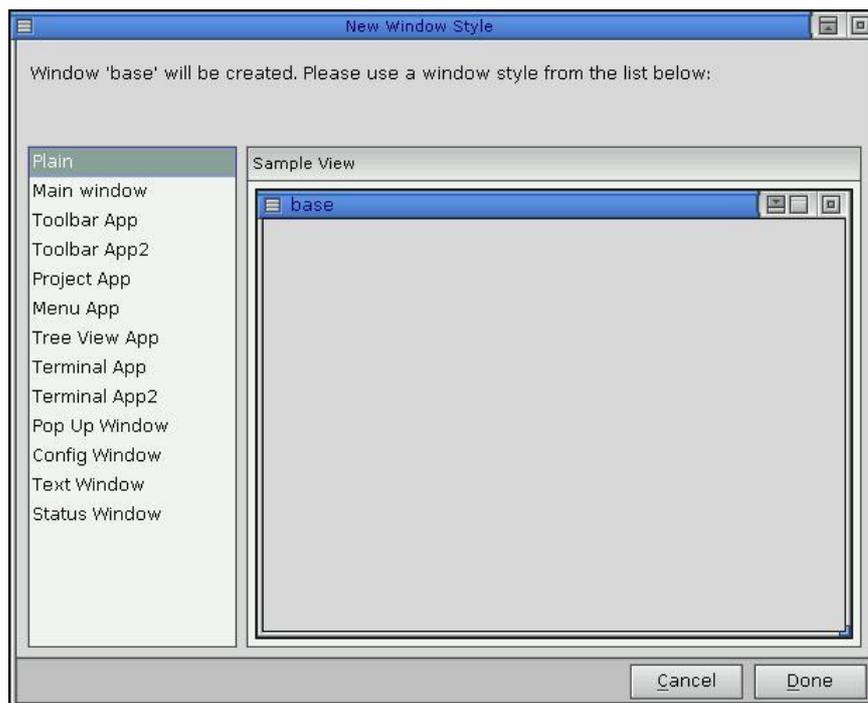


Рис. 3-1. Выбор стиля базового окна

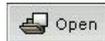
После выбора приложения Вам предстоит

- сохранить его, дав ему имя
- использовать диалог "Application Startup Information", чтобы
 - задать глобальный заголовочный файл
 - задать функцию инициализации
 - подключить или отключить опции командной строки.

- ☞ Вам стоит разработать соглашение по именам для всех виджетов, модулей, функций и прочая. Это облегчит управление Вашим приложением.

Открытие приложения

Чтобы открыть существующее приложение, выберите пункт "Open" из меню "File", нажав <Ctrl> + <O>, или выберите "Open" из панели инструментов PhAB:



Вы увидите переключатель приложений:

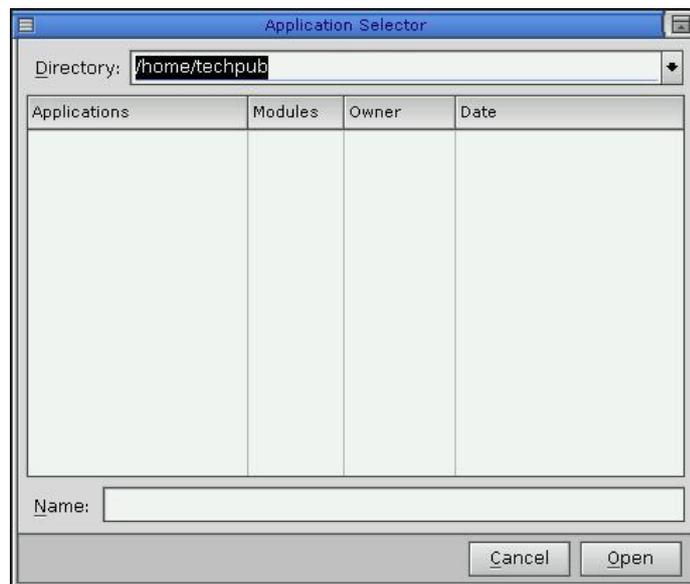


Рис. 3-2. Диалог выбора приложений

Если необходимое Вам приложение располагается в другой директории, наберите имя директории в области "Directory" и нажмите <Enter>. Чтобы выбрать приложение, сделайте одно из нижеследующего:

- Дважды щёлкните на приложении
или
 - щёлкните на приложении, затем нажмите <Enter> или щёлкните на "Open"
или
 - наберите имя приложения, затем нажмите <Enter> или щёлкните на "Open".
- ☞ Если кто-нибудь уже открыл это приложение, PhAB не будет его открывать, если только Вы только не запустили PhAB с опцией -n.
Если Вы используете NFS или SMB, Вы будете запускать PhAB с опцией -n, потому что иначе не сможете запереть файлы. Для получения более полной информации см. описание "appbuilder" в "Справочнике утилит QNX 6".

Сохранение приложения

Вы можете сохранить Ваше приложение несколькими способами, как это описано в разделе ниже.

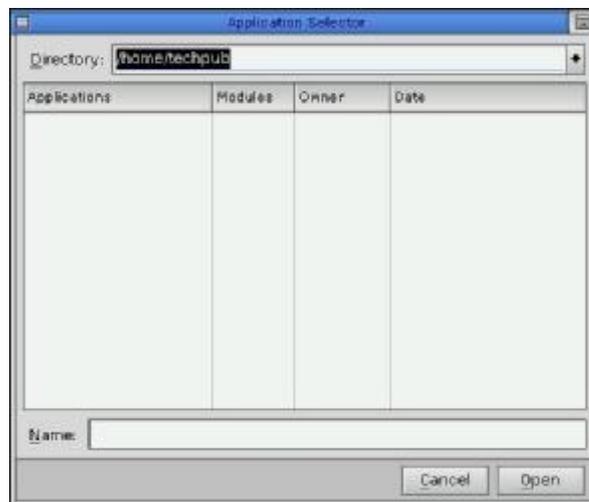
- ☞ Чтобы гарантировать действие самых последних изменений Вашего приложения, PhAB автоматически сохраняет Ваше приложение всякий раз, когда Вы регенерируете или собираете Ваше приложение.

Для получения информации об использовании ПО, обеспечивающего управление версиями, для приложений в PhAB, см. раздел "Управление версиями" в главе "Генерирование, компиляция и запуск программы на исполнение".

Именованное и переименование приложения

Чтобы сохранить новое безымянное приложение или сохранить приложение под другим именем или в другой директории:

1. Выберите пункт "Save As" из меню "File". Вы увидите диалог выбора приложения:



2. Диалог приводит список содержания директории. Если Вы хотите сохранить Ваше приложение в другой директории, наберите имя директории в области "Directory" и затем нажмите <Enter>.

- ☞ Если Вы наберёте новое имя директории, она сохранится. В следующий раз, когда Вы захотите посмотреть эту директорию, щёлкните на кнопке справа от области директории и выберите директорию из списка.

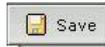
3. Наберите имя приложения в области "Name".
4. Нажмите <Enter> или щёлкните на "Save".

- ☞ Если Вы переименовали приложение, вы обнаружите, что имя исполняемого файла не переименовано. Это потому, что PhAB не изменил Makefile. Чтобы изменить имя исполняемого файла:

- Отредактируйте Makefile вручную и измените все вхождения имён исполняемых файлов или
- Если Вы не изменили Makefile, поскольку это была первая генерация, удалите его и регенерируйте приложение. См. главу "Генерирование, компиляция и запуск программы на исполнение".

Сохранение существующего приложения

Чтобы сохранить существующее приложение, выберите "Save" из меню "File", либо нажмите <Ctrl>+<S>, либо выберите кнопку "Select" на панели инструментов PhAB.



Переписывание существующего приложения

Чтобы переписать существующее приложение:

1. Выберите "Save As" из меню "File".
2. Сделайте одно из двух:
 - Дважды щёлкните на существующем приложении
или
 - Щёлкните на существующем приложении, затем нажмите <Enter> или щёлкните на "Save".

Заккрытие приложения

Чтобы закрыть приложение, выберите пункт "Close" из меню "File". Если Вы сделали какие-либо изменения, но не сохранили Ваше приложение, PhAB спросит Вас, желаете ли или нет сохранить эти изменения.

Задание стартовой информации приложения

Диалог задания стартовой информации приложения позволяет Вам задать типовые действия, выполняемые при запуске приложения.

Вы можете:

- включить или отключить опции командной строки
- определить глобальный хедер
- задать функцию инициализации
- включить имена экземпляров в виджеты
- указать, будет или нет генерироваться proto.h – см. раздел "Генерация прототипов функций" в главе "Генерирование, компилирование и запуск программы на исполнение"
- определить, какое окно должно появиться, когда приложение стартует.

Чтобы открыть этот диалог:

- выберите из меню "Application" пункт "Startup Info/Modules"
или
- Нажмите <F2>

Вот диалог, с некоторой примерной информацией, которой он заполнен:

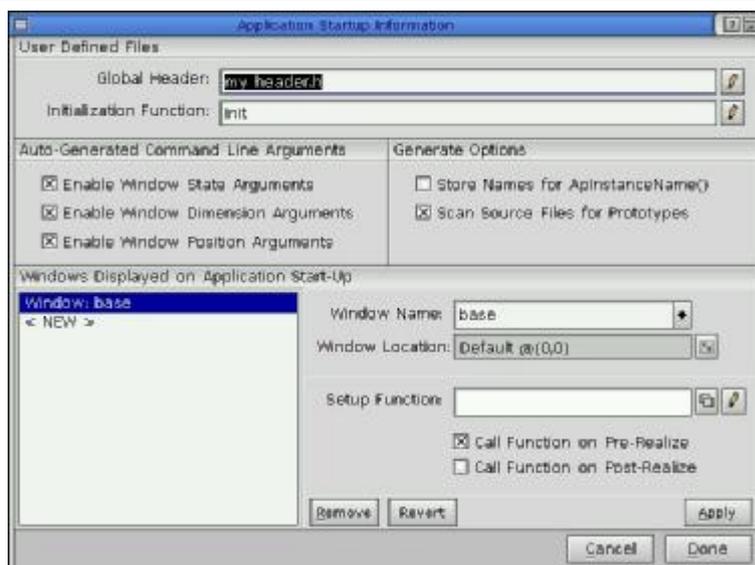


Рис. 3-3. Диалог стартовой информации приложения

После того, как Вы сделаете Ваши изменения, щёлкните на "Done".

Задание глобального заголовочного файла

Большинство приложений имеет глобальный хедер, включающий все файлы исходников. Если Вы планируете использовать в Вашем приложении глобальный хедер, Вы можете его задать перед тем, как PhAB сгенерирует какой-либо код. Это позволит PhAB автоматически включить хедер в каждый генерируемый им файл.

Чтобы задать глобальный хедер:

1. Нажмите <F2> или выберите пункт "Startup Info/Modules" из меню "Application". Вы увидите диалог "Application Startup Information".
2. В области "Global Header" наберите имя файла, который Вы собираетесь использовать. Вам не надо включать расширение ".h".

Например, чтобы задать хедер-файл `globals.h`, Вы можете просто ввести `globals`.

3. Чтобы немедленно редактировать хедер, щёлкните на иконке рядом с областью "Global Header". Вы можете редактировать хедер только если Вы присвоили имя приложению, сохраняя его. Формат хедер-файла обсуждается в главе "Работа с кодом".

☞ Если Вы задали хедер после того, как какой-то код уже был сгенерирован, Вы должны вернуться назад и вручную добавить хедер к файлам, которые уже сгенерированы.

Функция инициализации

Ваше приложение может включать функцию инициализации, вызываемую перед тем, как будут созданы какие-либо модули или виджеты. В ней Вы можете инициализировать данные, открыть базу данных виджетов, установить обработчики прерываний и прочая. Чтобы задать функцию инициализации:

1. Нажмите <F2> или выберите пункт "Startup Info/Modules" из меню "Application". Вы увидите диалог "Application Startup Information".
2. В области "Initialization Function" наберите имя функции инициализации.

Когда Вы задаёте функцию инициализации, PhAB генерирует заготовку функции, чтобы получить информацию о том, как задавать язык (C или C++) и имя файла, см. раздел "Имена функций и файлов" в главе "Работа с кодом".

3. Чтобы немедленно отредактировать функцию, щёлкните на иконке рядом с областью "Initialization Function". Вы можете редактировать функцию, только если Вы присвоили имя приложению, сохраняя его. Прототип этой функции обсуждается в главе "Работа с кодом".

Опции командной строки

По умолчанию все сгенерированные в PhAB приложения имеют следующие опции командной строки:

-h height[%] " height " – это высота окна в пикселях или как процент от высоты экрана, если задано "%".

-s server_name " server_name " – имя сервера Photon:

ЕСЛИ SERVER_NAME:	ИСПОЛЬЗУЕТСЯ ЭТОТ СЕРВЕР:
node_path	node_path/dev/photon
fullpath	fullpath
relative_path	/dev/relative_path

-w width[%] "width" – это ширина окна в пикселях или как процент от ширины экрана, если задано "%".

-x position [%] [r] координата по оси x верхнего левого угла окна в пикселях или в % от ширины экрана, если задано "%". Если задано "r", координата является относительной – от текущей консоли.

-y position [%] [r] координата по оси y верхнего левого угла окна в пикселях или в % от высоты экрана, если задано "%". Если задано "r", координата является относительной – от текущей консоли.

-Si|m|n состояние инициализации основного окна (свёрнутое в иконку, максимизированное или нормальное).

По умолчанию все эти опции включены, так что пользователь может динамически перемещать или изменять размер приложения или задавать его состояние инициализации. Например, чтобы запустить приложение на 4-й консоли (центр рабочего пространства), задайте опцию командной строки:

-x 100% -y 100%

API PhAB'a обработает эти опции перед вызовом функции инициализации; если Вы планируете добавить к приложению Ваши собственные опции командной строки, убедитесь, что Вы выбрали опции, которые не конфликтуют с этими. Вам следует также написать код обработки опций для обработки и игнорирования этих опций. Если Вы не сделаете этого, то увидите сообщение об ошибке на консоли, когда запустите приложение на исполнение. См. обсуждение функции инициализации в главе "Работа с кодом".

Если Вы не хотите позволить пользователю перемещать или изменять размер приложения:

1. Нажмите <F2> или в меню "Application" выберите пункт "Startup Info/Modules", чтобы открыть диалог "Application Startup Information".
2. Установите кнопки переключения для опций, как Вы желаете.

Включение имён экземпляров

PhAB преобразует имена экземпляров Ваших виджетов в декларации ABN_..., так что Вы можете использовать их в тексте своей программы для ссылки на Ваш виджет по имени. Вы можете при желании включить текстовую строку с именем экземпляра в память виджета. Чтобы сделать это:

1. Нажмите <F2> или в меню "Application" выберите пункт "Startup Info/Modules", чтобы открыть диалог "Application Startup Information".
2. Щёлкните на кнопке "Store Names for ApInstanceName()".

☞ Включение имён экземпляров увеличивает объём памяти, требуемой для запуска Вашего приложения. Чтобы найти эту строку для виджета, используйте функцию ApInstanceName() – см. "Справочник библиотеки Photon" для получения большей информации.

Окна запуска

Когда Вы впервые создаёте приложение, в качестве начального окна и только как окна запуска устанавливается обязательное базовое окно. Используя диалог "Application Startup Information", Вы можете указать Вашему приложению:

- использовать другое окно как начальное окно запуска
- отображать как окна запуска несколько окон
- не использовать окно запуска.

Окно, появляющееся первым в списке открытых/стартующих окон, является начальным окном запуска:

- это первое окно, которое будет отображено
- это действует как принимаемое по умолчанию родительское окно для всех других окон и диалогов
- его закрытие означает завершение приложения.

Обычно главное окно приложения является первым создаваемым окном.

Для каждого окна в списке запуска Вы можете задать информацию, которая идентична используемой при создании ответной реакции модульного типа, как описано в главе "Редактирование ресурсов и ответных реакций в PhAB". Информация по каждому окну включает:

Window Name	Имя модуля окна. Для выбора из списка существующих окон щёлкните на иконке рядом с этой областью. Если Вы задали имя несуществующего модуля, PhAB спросит, желаете ли Вы создать этот модуль.
Window Location	Место, где появится окно; см. раздел "Позиционирование модуля" в главе "Работа с модулями".
Setup Function	Функция, вызываемая при реализации окна (необязательная). Для редактирования этой функции щёлкните на иконке рядом с этой областью. Кнопки под именем функции определяют, будет ли функция вызываться до реализации окна, после того как окно реализовано, или в обоих случаях.
Apply	Выполнить внесённые изменения.
Revert	Восстанавливает информацию об окне в первоначальном состоянии.
Remove	Удаляет выбранные окна из списка запуска.

Добавление окна запуска

Чтобы добавить новое окно в список окон запуска, щёлкните на <NEW>, заполните информацию по окну и щёлкните на "Apply".

Модификация окна запуска

Для модификации существующего окна запуска выберите окно из списка "Windows Opened/Startup", введите все требуемые Вами изменения в области информации окна и затем щёлкните на "Apply".

Удаление окна запуска

Чтобы удалить окно запуска, выберите окно из списка "Windows Opened/Startup" и щёлкните на "Remove".

Импортрование файлов

PhAB позволяет Вам импортировать несколько типов файлов, используя пункт "Import Files" в меню "File":

- модули PhAB

- файлы графических образов.

Шаги одинаковы для всех типов:

1. Выберите пункт "Import Files" из меню "File", затем выберите соответствующий тип из подменю "Import Files". Вы увидите диалог, позволяющий выбрать файлы – файловый селектор.
2. Файловый селектор отображает доступные файлы заданного типа в текущей директории.
3. Для выбора файла выполните одно из следующих действий:
 - дважды щёлкните по файлу
 - щёлкните по файлу, затем нажмите <Enter> или щёлкните на "Open":
 - наберите имя файла, затем нажмите <Enter> или щёлкните на "Open".

Импортowanie модулей PhAB из других приложений

При импортowaniu модулей PhAB из других приложений файловый селектор может отображать несколько модулей. Каждый тип модуля имеет своё расширение; см. раздел "Типы модулей" в главе "Работа с модулями".

- ☞ Ответные реакции не могут быть импортрованы, импортуются только сами модули и виджеты. После импортowania модуля Вы можете прикрепить новые специфические для приложения ответные реакции. Обычно PhAB помнит имя экземпляра каждого импортванного виджета. Однако, если он обнаруживает дублирование имени, то изменяет это имя на имя класса виджета, чтобы избежать ошибки генерации кода.

Импортowanie графических образов

При импортowaniu графических модулей файловый селектор отображает все файлы со следующими расширениями:

- .bmp
- .gif
- .jpg
- .psx
- .xbm

PhAB импортует графику как PtLabel в текущий выбранный модуль и устанавливает ресурс виджета Pt_ARG_LABEL_TYPE в значение Pt_IMAGE. Если Вы хотите редактировать импортванный образ, используйте попиксельный редактор, как описано в главе "Редактирование ресурсов и ответных реакций в PhAB".

Глава 4. Работа с модулями

Модули – это основные составляющие пользовательского интерфейса приложения PhAB. Эта глава описывает, как работать с ними, и включает:

- Типы модулей
- Анатомия модулей
- Выбор модуля
- Как хранятся модули
- Изменение ресурсов модулей
- Использование селектора модуля
- Создание нового модуля
- Просмотр модуля
- Открытие модуля
- Удаление модуля
- Сворачивание модуля в иконку
- Закрывание модуля
- Отображение модулей в реальном времени
- Поиск потерявшихся модулей и иконок
- Модули окна
- Модули диалога
- Модули меню
- Модули картинки
- Модули иконки

Модули служат контейнерами, содержащими виджеты Вашего приложения. Некоторые модули, такие как окна или диалоги, являются в действительности виджетами контейнерного класса и дают Вам возможность размещать виджеты непосредственно в них. Другие, такие как иконки и меню, имеют либо предопределённые виджеты, либо специализированный редактор для создания виджетов, которые они содержат.

Типы модулей

PhAB предлагает ряд типов модулей, каждый для специфического использования. Тип модуля идентифицируется по:

- панелям управления, если модуль выбран
- иконке, если модуль минимизирован
- расширению файла, создаваемого PhAB для модуля при генерации кода приложения.

! Файлы модулей являются бинарными, не редактируйте их текстовым редактором или Вы можете их повредить.

МОДУЛЬ	ИСПОЛЬЗОВАНИЕ	РАСШИРЕНИЕ
Window	Главные функции приложения	.wgtw
Dialog	Получение дополнительной информации от пользователя	.wgt d
Menu	Многоуровневые текстовые меню	.wgt m
Picture	Изменение содержания существующего модуля или создание базы данных виджета	.wgt p
Icon	Иконки Вашего приложения для использования в системных панелях (shelf) и кнопках запуска (launcher plugins)	.wgt i

Анатомия модуля

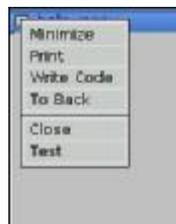
PhAB отображает каждый модуль как окно в своей рабочей области. Как и окна, модули имеют набор кнопок управления на своих рамках.



Анатомия типичного модуля PhAB

Большинство модулей включают такие элементы:

- Кнопка рабочего меню
Вызывает рабочее меню модуля



Рабочее меню модуля

Рабочее меню включает

- Minimize – свернуть модуль в иконку
- Print – печатать модуль
- Write Code – генерировать код для модуля
- To Back – перевести этот модуль за другие модули в рабочей области PhAB
- Close – закрыть модуль

Заметьте, что использование рабочего меню для закрытия модуля не удаляет модуль. Он просто удаляется из рабочей области, что даёт Вам больше свободного места для работы в ней. Для того, чтобы в любой момент вернуть модуль назад, используйте селектор модулей, который позволяет Вам получить доступ, создать и удалить любой тип модуля. Для получения более полной информации см. раздел "Использование селектора модулей" в этой главе.

- Test
Переключает в режим тестирования, так что Вы можете провзаимодействовать с виджетами, как будто-бы запущено приложение.
Вы можете переключиться **в** и **из** режима тестирования с помощью кнопки "Test" на инструментальной панели PhAB (см. раздел "Инструментальные панели" в главе об окружении PhAB) или кнопки "Test" на заголовочной панели модуля.
 - Title bar
(заголовочная панель)
Отображает имя экземпляра модуля. Чтобы перемещать модуль, укажите на эту панель и перетаскивайте курсор.
 - Collapse button
Сворачивает модуль просто в заголовочную панель.
 - Minimize button
Сворачивает модуль в иконку.
 - Test button
Аналогична пункту "Test" в рабочем меню, она позволяет Вам переключать модуль в режим тестирования.
 - Close button
Закрывает модуль
- (только для определённых модулей)

Выбор модуля

Чтобы выбрать модуль, находящийся на рабочей области PhAB:

- Щёлкните на заголовочной панели модуля
или
- если модуль свёрнут в иконку, дважды щёлкните по его иконке
или
- Вызовите меню "Window" и выберите модуль по имени (это работает как для свёрнутых в иконку модулей, так и для несвёрнутых).

Какой бы метод Вы ни избрали, Вы увидите элементы управления размером – признак того, что модуль выбран.

Чтобы узнать, как открывается модуль, который не находится в рабочей области PhAB, см. раздел "Открытие модуля" в этой главе.

Как сохраняются модули

Когда Вы сохраняете Ваше приложение, PhAB сохраняет все модули приложения как файлы в директории приложения wgt. Каждый модуль сохраняется в своём собственном файле с расширением файла, основанном на типе модуля. Позже, когда Вы делаете "make" для своего приложения, PhAB связывает все модули в исполняемый бинарный файл. Это делает приложение одной автономной программой, которую Вы можете легко распространять.

Для получения более полной информации см. раздел "Как организованы файлы приложения" в главе "Генерирование, компилирование и запуск программы на исполнение".

Изменение ресурсов модуля

Когда Вы выбираете модуль в PhAB, панель управления ресурсами изменяется, отображая список ресурсов виджета, доступных для этого класса модулей. В зависимости от того, какой ресурс Вы изменяете, Вы можете не увидеть немедленного эффекта. Все сделанные изменения возымеют, однако, эффект при запуске приложения.

Поскольку PhAB отображает все модули как порождённые окна внутри рабочей области, Вы можете работать с любым числом модулей одновременно.

Использование селектора модулей

Когда Вы используете меню "Application" для создания или просмотра модуля PhAB любого типа (окна, диалога, меню, прочая), PhAB отображает диалог селектора модулей:

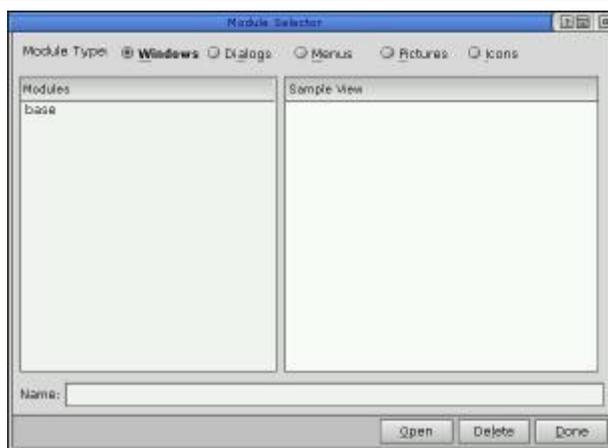


Рис. 4-1. Диалог селектора модулей

В верхней части диалога Вы увидите набор радиокнопок. Они позволяют Вам выбрать, какой тип модуля Вы хотите создать/открыть, удалить или закрыть. Принимаемым по умолчанию типом является тип, определённый пунктом меню, который Вы выбрали при вызове диалога.

Создание нового модуля

Чтобы создать любой новый модуль, следуйте этим простым шагам:

1. Из меню "Application" выберите тип модуля, который Вы хотите создать. Вы увидите селектор модулей.
2. В области "Name" наберите имя экземпляра объекта для нового модуля, затем нажмите <Enter> или щёлкните на "Open".
 - ☞ Имя модуля не должно быть более чем в 48 символов и должно быть применимо к идентификаторам языка C. Если Вы создаёте модуль иконки к основному окну Вашего приложения, Вы должны присвоить модулю иконки имя. Это то имя, которое приложение может использовать для получения иконки.
3. Для модулей окна и диалога PhAB предлагает Вам выбрать стиль из диалога, отображающего возможные варианты выбора. Для других типов модулей PhAB просто спрашивает, создавать или нет новый модуль. Нажмите <Enter> или щёлкните "Yes". Вы увидите новый модуль в рабочей области PhAB.
4. Щёлкните на "Done".

Для более полной информации по созданию конкретных типов модулей см. раздел по каждому типу модуля в этой главе. Вы можете также импортировать модули из других приложений PhAB. Для получения более полной информации см. раздел "Импортирование файлов" в главе "Работа с приложениями".

Просмотр модулей

Чтобы просто просмотреть любой модуль:

1. В меню "Application" выберите тип модуля, который Вы хотите увидеть. Вы увидите селектор модуля.
2. Щёлкните на имени модуля в скроллируемом списке модулей.

Открытие модулей

Чтобы отобразить модуль в Вашей рабочей области, выберите его из списка модулей в меню "Window" или выполните следующее:

1. Из меню "Application" выберите тип модуля, который Вы хотите открыть. Вы увидите селектор модулей.
2. Выполните следующее:
 - дважды щёлкните на имени модуля
или
 - щёлкните на имени модуля, затем нажмите <Enter> или щёлкните на "Open"
или
 - наберите имя модуля, затем нажмите <Enter> или щёлкните на "Open".

Удаление модуля

Чтобы удалить модуль:

1. В меню "Application" выберите тип модуля, который хотите удалить. Вы увидите селектор модулей.
2. Щёлкните на имени модуля.
3. Щёлкните на кнопке "Delete".

☞ Удаление модуля не приводит к удалению файла модуля; при этом просто удаляется имя из списка. Все ответные реакции, принадлежащие модулю или его потомкам, удаляются.

Сворачивание модулей в иконки

Рабочая область PhAB позволяет Вам работать одновременно с несколькими модулями приложения. Вы можете свернуть модули в иконки для формирования Вашей рабочей области. Чтобы свернуть любой модуль на рабочей области в иконку:

- Дважды щёлкните на кнопке рабочего меню модуля (верхний левый угол модуля)
или
- щёлкните на кнопке рабочего меню и выберите "Minimize"
или
- щёлкните на кнопке минимизации на заголовочной панели модуля.

Свернувшись в иконку, модуль располагается в нижней части рабочей области. Вы можете перетащить его в любое место рабочей области, так, например, Вы можете сгруппировать вместе иконки модулей, которые вместе используются или связаны.

Заккрытие модуля

Если Вы желаете остановить работу модуля или удалить его из рабочей области:

1. Щёлкните на кнопке рабочего меню модуля (верхний левый угол модуля).
2. Выберите пункт "Close".

Если Вы хотите просто свернуть модуль в иконку, выберите пункт Minimize".

Чтобы вновь открыть какой-либо закрытый модуль, см. раздел "Селектор модулей" в этой главе.

Отображение модулей в реальном времени

Для Вашего приложения может понадобиться отобразить модуль во время его работы. Вы можете:

- Создать виджет, использующий ответную реакцию, чтобы отображать модуль. Например, Вы можете создать `PtButton` с ответной реакцией модульного типа, которая отображает модуль. Для получения более полной информации см. раздел "Редактирование ответных реакций" в главе "Редактирование ресурсов и ответных реакций в PhAB".
- Использовать внутренний линк (`internal link`) для создания модуля в тексте программы Вашего приложения. См. главу "Доступ к модулям PhAB из программы".

Позиционирование модуля

Вы можете задать, где отображать модуль, когда Вы создаёте связывающую ответную реакцию от виджета к этому модулю. Чтобы выполнить это, используйте диалог расположения – "Location dialog".

Чтобы открыть диалог расположения и выбрать расположение модуля:

1. При создании или редактировании ответной реакции, связанной с модулем, щёлкните на области "Location" или на иконке справа от области. Вы увидите список расположений:



Рис. 4-2. Диалог расположения

Для окон и диалогов принимаемым по умолчанию расположением является (0, 0), при котором окно располагается на следующей доступной позиции, определяемой менеджером окон. Для модулей меню принимаемым по умолчанию расположением является "Below Widget" – под виджетом.

2. Щёлкните на желаемом расположении.
3. Вы можете также задать смещение x и y . Например, если Вы устанавливаете расположение в нижнем правом углу и устанавливаете смещение по x на -100 , окно будет отображаться так, что его нижний правый угол будет на 100 пикселей левее нижнего правого угла экрана.

☞ Если Вы выберете расположение по умолчанию, смещения игнорируются.

4. Щёлкните на "Done".

Нахождение потерянных модулей и иконок

Чтобы найти потерянный модуль или иконку:

- Выберите из меню "Window" пункт "Arrange Modules". PhAB расположит все открытые модули каскадом в рабочей области PhAB.
- Выберите из меню "Window" пункт "Arrange Icons". PhAB расставит все существующие иконки вдоль нижней кромки рабочей области.

Если вышеприведенные приёмы не сработали, Вы можете "закрыть" модуль. (PhAB позволяет Вам закрывать модуль, удаляя его из рабочей области и уменьшая толкучку). Чтобы узнать, как вновь открыть закрытый модуль, см. раздел "Открытие модуля" в этой главе.

Модули окон

КЛАСС ВИДЖЕТА	РАСШИРЕНИЕ ФАЙЛА	СОЗДАНИЕ ВИДЖЕТА
PtWindow	.wgtw	Непосредственно из палитры виджетов

Обычно Вы используете модули окон как основные активные сущности Вашего приложения. Так как большинство приложений использует модуль окна для своего основного окна, PhAB автоматически генерирует модуль окна с именем "base", когда Вы впервые создаёте какое-либо приложение. Он также предусматливает информацию запуска приложения, делая базовое окно открывающимся при запуске приложения. (См. раздел "Задание информации запуска приложения" в главе "Работа с приложениями").



Иконка модуля окна

Модули окон могут поддерживать множественность экземпляров. Это означает, что одновременно может отображаться две и более копий одного и того же модуля окна. В результате Вам необходимо хранить жизненный путь каждого указателя на экземпляр окна, генерируемого при создании окна. Благодаря этому Вы всегда знаете, с каким окном имеете дело, когда обрабатываете ответную реакцию. Для получения более полной информации см. раздел "Обработка множественных экземпляров окна" в главе "Работа с кодом".

Даже несмотря на то, что базовое окно Вашего приложения является модулем окна, Вы обычно отображаете его только однажды, при запуске. Так что пока Вашему приложению не понадобится отобразить более одной копии базового окна одновременно, Вы не храните трассировку указателя на экземпляр базового окна.

Как пример кода обработки множества экземпляров модулей окна см. раздел "Создание окон" в главе "Уроки".

Изменение размера модуля окна

Когда Вы устанавливаете в PhAB размеры модуля окна, это и будут его размеры при запуске приложения.

Модули диалога

КЛАСС ВИДЖЕТА	РАСШИРЕНИЕ ФАЙЛА	СОЗДАНИЕ ВИДЖЕТА
PtWindow	.wgt	Непосредственно из палитры виджета

Модули диалога позволяют Вам получать дополнительную информацию от пользователей. Обычно Вы используете эту информацию для осуществления конкретной команды или задачи.



Иконка модуля диалога

Большинство модулей диалогов включают следующие кнопки:

- "Done" – позволяет пользователю указать, что он завершил ввод информации
- "Cancel" или "Close" – позволяют пользователю закрыть диалог без ответа.

С точки зрения PhAB модули диалогов почти идентичны модулям окна, с одним важным отличием – модуль диалога может иметь только один активный экземпляр. Так что если Вы вызываете диалог, который уже открыт, API PhAB просто выводит существующий экземпляр диалога на передний план. Это свойство связано с природой диалога – Вам редко когда может понадобиться вводить одну и ту же информацию дважды. Если несмотря на все соображения, Вам потребуется диалог, который мог бы поддерживать множественность экземпляров, используйте модуль окна.

Ограничение диалога одним экземпляром упрощает обработку ответной реакции, поскольку Вы можете использовать декларации виджета, генерируемые PhAB, для получения доступа к виджетам внутри диалога. Для получения более полной информации см. раздел "Имена экземпляров" в главе "Создание виджетов в PhAB".

Изменение размеров модуля диалога

Когда Вы устанавливаете размеры модуля диалога в PhAB, это и будут его размеры при запуске приложения.

Предопределённые диалоги

Библиотеки Photon включают удобные функции, определяющие различные полезные диалоги:

ApError()	Отображает диалог сообщения об ошибке
PtAlert()	Отображает сообщение и требует подтверждения
PtFileSelection()	Создаёт диалог выбора файла
PtFontSelection()	Создаёт диалог выбора шрифта
PtMessageBox()	Вызывает всплывающее табло с сообщением
PtNotice()	Отображает сообщение и ждёт уведомления
PtPassword()	Запрашивает пароль
PtPrintPropSelect()	Изменяет опции печати для выбранного принтера через модальный диалог
PtPrintSelect()	Отображает модальный диалог настройки для выбора опций печати
PtPrintSelection()	Отображает модальный диалог для выбора опции настройки
PtPrompt()	Отображает сообщение и получает текстовый ввод от пользователя

Модули меню

КЛАСС ВИДЖЕТА	РАСШИРЕНИЕ ФАЙЛА	СОЗДАНИЕ ВИДЖЕТА
PtMenu	.wgtm	Специальный редактор

Модуль меню предоставляет многоуровневое текстовое меню. В отличие от остальных модулей, модуль меню не позволяет Вам создавать виджеты непосредственно внутри него. Вместо этого Вы используете редактор меню PhAB для создания пунктов меню.



Иконка модуля меню

Открытие редактора меню

Чтобы открыть редактор меню:

1. Выберите модуль меню
2. Щёлкните на "Menu Items" на панели управления ресурсами, и PhAB отобразит редактор меню:

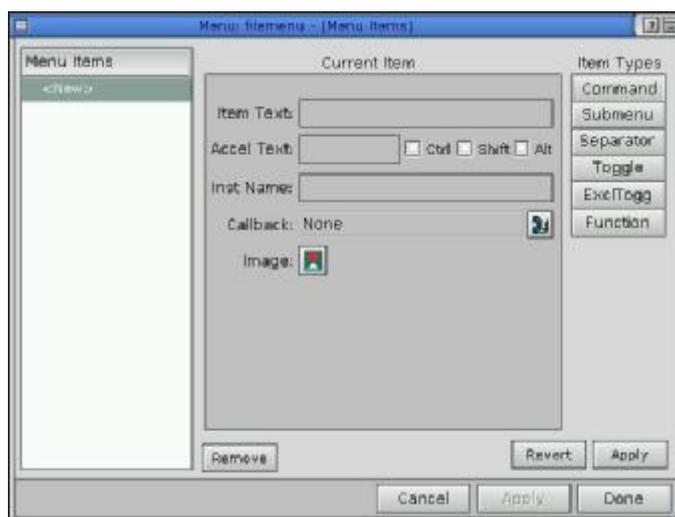


Рис. 4-3. Редактор меню PhAB

В верхнем правом углу Вы можете видеть кнопки, представляющие типы создаваемых Вами меню:

- Command – вызывает ответную реакцию PhAB
- Submenu – отображает порождённое меню
- Separator – вставляет линии или разрядку между другими пунктами меню
- Toggle или Excl Togg (эксклюзивный переключатель) – изменяет или отображает состояние приложения
- Function – задаёт функцию приложения, которая может динамически добавлять пункты меню в меню.

Эти кнопки расположены в нижней части диалога:

КОГДА ВЫ ХОТИТЕ:	ИСПОЛЬЗУЙТЕ КНОПКУ:
Принять все изменения и закрыть редактор	Done
Принять все изменения и продолжить редактировать меню	Apply
Отменить все изменения, сделанные с момента открытия редактора	Cancel

Задание имён экземпляров

Чтобы создать любой командный или переключаемый пункт меню (т.е. любой пункт, который может вызывать ответную реакцию), Вы должны ввести уникальное имя экземпляра – этого требует PhAB. Имя экземпляра позволяет Вам получить доступ к пунктам меню из программного кода Вашего приложения.

Когда PhAB генерирует код Вашего приложения, он генерирует глобальную переменную `ABN_...` для каждого пункта меню, который требует этого. Вы используете эту переменную в функциях API, связанных с пунктами меню – `ApModifyItemState()` и `ApModifyItemText()`.

Например, пусть, скажем, пункт меню не должен быть доступен, когда пользователь щёлкает на виджете, вызывающем меню. Используя имя экземпляра, Вы можете сделать тусклым этот пункт перед тем, как отображать меню. Для получения более полной информации см. "Инициализация меню" в главе "Работа с программным кодом".

Создание "горячих клавиш" и "кнопок быстрого доступа" (hotkeys and shortcuts)

Чтобы помочь пользователю выбрать пункт меню более быстро, Вы можете:

- обеспечить клавиатурную кнопку быстрого доступа, по которой бы выбирался пункт
- обеспечить горячую клавишу, которая бы напрямую вызывала команду, представленную пунктом меню, даже когда меню невидимо.

Клавиатурная кнопка быстрого доступа работает только когда меню в настоящий момент видимо. С другой стороны, горячая клавиша работает вне зависимости от того, видимо или нет меню. Создание кнопки быстрого доступа просто. Когда Вы вводите "Item Text", просто поместите символ "&" перед символом, который будет действовать как кнопка быстрого доступа. Например, пусть мы создаём пункт "Save As". Вы можете ввести "Save &As", результатом чего будет подчёркнутая буква "A". Когда меню открывается, пользователь может нажать либо "A" либо вызвать ответную реакцию, связанную с "Save As".

Создание горячей клавиши вынуждает затратить ненамного больше труда, но так же легко делается. Во-первых, Вы должны убедиться, что горячая клавиша появилась рядом с пунктом меню, когда меню отображается. Чтобы это сделать, используйте область "Accel Text". Например, пусть скажем, горячей клавишей для пункта меню "Save" будет `<Ctrl>+<S>`. В этом случае Вы должны ввести "S" в области "Accel Text" и включить переключатель "Ctrl".

Далее, Вам необходимо создать ответную реакцию горячей клавиши для сочетания `<Ctrl>+<S>`. Поскольку меню может быть не создано на момент, когда пользователь нажал `<Ctrl>+<S>`, Вы не можете прикрепить ответную реакцию горячей клавиши к меню или к пункту меню. Вместо этого Вы должны прикрепить ответную реакцию к главному модулю приложения, которым обычно является модуль базового окна. При задании функции ответной реакции горячей клавиши, используйте ту же самую функцию, которую Вы определили для ответной реакции пункта меню.

Если по каким-либо причинам Вам необходимо различать два метода вызова ответной реакции, необходимо проверить код аргумента ответной реакции. Горячие клавиши всегда имеют код аргумента `Pt_CB_HOTKEY`.

Для получения более полной информации по созданию ответных реакций горячих клавиш см. раздел "Ответные реакции горячих клавиш" в главе "Редактирование ресурсов и ответных реакций в PhAB".

Изменение размеров модуля меню

Вы можете чувствовать себя совершенно свободными в возможностях изменять размеры меню для того, чтобы сделать их легче читаемыми или же для уменьшения занимаемого пространства. Когда Вы запускаете приложение, реальные размеры виджета `PtMenu` будут определяться пунктами меню.

Создание командных пунктов меню

Командные пункты меню позволяют Вам вызывать код приложения или отображать модуль.

ОБЛАСТЬ	ОПИСАНИЕ
Item Text	Текст, который будет отображаться
Accel Text	Горячая клавиша для вызова команды
Inst Name	Имя, используемое внутри программного кода приложения
Callback	Функция, которая будет вызвана, когда пункт выбран
Image	Иконка для использования в пункте меню

Чтобы создать командный пункт меню:

1. Щёлкните на <NEW>
2. Щёлкните на кнопке "Command" в верхнем правом углу
3. В области "Item Text" введите текст пункта. Чтобы создать кнопку быстрого доступа, разместите символ "&" перед символом, который будет действовать как кнопка быстрого доступа. Например, скажем, Вы ввели &File. В этом случае пользователь может выбрать пункт, нажав <F>.
4. В области "Inst Name" введите имя экземпляра, которое Вы будете использовать.
5. Если Вы планируете иметь ответную реакцию горячей клавиши для этого пункта, введите строку горячей клавиши и модификатор клавиши (например, <Ctrl>+<S>) в области "Accel Text". Горячая клавиша отображается в меню как напоминание для пользователя.
6. Добавьте ответную реакцию PhAB, щёлкнув на иконке ответной реакции: 

Чтобы получить более полную информацию, см. раздел "Редактирование ответных реакций" в главе "Редактирование ресурсов и ответных реакций в Phab".

7. Добавьте, если считаете целесообразным, картинку в виде образа.
8. Щёлкните на "Apply", чтобы добавить пункт в меню.

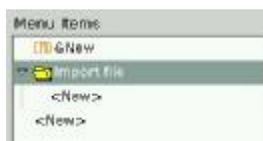
Создание пунктов подменю

Пункты подменю позволяют Вам создавать другие уровни меню.

ОБЛАСТЬ	ОПИСАНИЕ
Item Text	Текст, который будет отображаться
Inst Name	Имя, используемое внутри программного кода приложения

Чтобы создать пункт подменю:

1. Щёлкните на <NEW>
2. Щёлкните на кнопке "Submenu" в верхнем правом углу.
3. В области "Item Text" наберите имя подменю. Чтобы создать клавиатурную кнопку быстрого доступа, разместите символ "&" перед символом, который будет действовать как кнопка быстрого доступа (точно так же, как для командных пунктов меню выше).
4. Щёлкните на "Apply".
5. Список "Menu Items" отобразит подменю:



6. Вы можете теперь добавить пункты в подменю, выбрав <NEW> в подменю.

Создание разделителей пунктов

Разделитель пунктов позволяет Вам создавать интервалы между пунктами меню. Это может оказаться удобным при его использовании для создания логических групп пунктов меню. Вы можете выбрать из нескольких стилей разделителей.

Для создания разделителя меню:

1. Щёлкните на <NEW>
2. Щёлкните на кнопке <Separator> в верхнем правом углу.

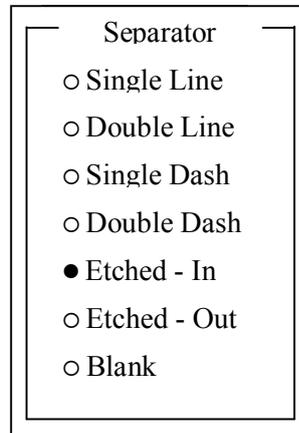


Рис. 4-4. Стили разделителя меню

Вы увидите список стилей разделителя, которые означают соответственно:

- Одинарная линия
- Двойная линия
- Одинарная пунктирная линия
- Двойная пунктирная линия
- "Выгравленная" внутрь
- "Выгравленная" наружу
- Пустая линия

3. Щёлкните на стиль разделителя, который Вам нужен, затем щёлкните на "Apply".

Создание пунктов-переключателей

Пункты-переключатели позволяют Вам изменять или отображать некие состояния приложения, которые могут быть включены или выключены.

ОБЛАСТЬ	ОПИСАНИЕ
Item text	Текст, который будет отображаться
Inst Name	Имя, используемое внутри программного кода приложения
Callback	Функция, которая будет вызвана, когда пункт выбран
Image	Иконка для использования в пункте меню

Чтобы создать пункт-переключатель:

1. Щёлкните на <NEW>, затем щёлкните на кнопке "Toggle".
2. Следуйте той же процедуре, что при создании командных пунктов меню.

Создание пунктов-функций

Пункт-функция позволяет Вам задать функцию приложения, которая динамически добавляет пункты меню в меню во время исполнения программы. Например, Вы можете использовать пункт меню в меню "File", чтобы отобразить последние три файла, с которыми работал пользователь.

Библиотека PhAB вызывает заданную функцию как встроенную в меню. Динамически созданные пункты меню появляются там, где Вы расположили пункт-функцию в меню.

ОБЛАСТЬ	ОПИСАНИЕ
Function	Функция, которая будет вызываться

Чтобы создать пункт-функцию:

1. Щёлкните на <NEW>, затем щёлкните на кнопке "Function".
2. В области "Function" введите имя функции приложения, которая будет динамически добавлять пункты-меню в меню.
Если Вы зададите это имя функции, PhAB сгенерирует заготовку функции; для получения более полной информации, специфической для языка (C или C++) и имени файла, см. раздел "Имена функций и имена файлов" в главе "Работа с программным кодом".
3. Вы можете сразу же отредактировать функцию, щёлкнув на кнопке справа от имени функции.
4. Щёлкните на "Apply".

Для получения более полной информации по функциям приложения см. раздел "Генерирование пунктов меню" в главе "Работа с программным кодом".

Перемещение пунктов меню

Скроллируемый список "Menu Items" позволяет Вам перемещать пункты в меню на другую позицию в меню. Пусть, скажем, Вы хотите переместить пункт по имени "Browse", так чтобы он появлялся непосредственно перед пунктом по имени "Edit". Вы должны:

1. Перетащить пункт "Browse", пока его контур не окажется непосредственно над пунктом "Edit".
2. Отпустить кнопку мыши. Пункт "Browse" появится в новой позиции.

Использование модуля меню

Как только Вы создали модуль меню, Вы должны сделать так, чтобы Ваше приложение его отображало. Обычно Вы должны выполнить следующее:

1. Создайте PtMenuBar в верхней части окна.
2. Добавьте PtMenuButton к панели меню, присвоив ему соответствующие имя экземпляра и текстовую строку.
3. Добавьте ответную реакцию связи с модулем к списку ответных реакций Pt_CB_ARM кнопок меню.

☞ Вы можете добавить ответную реакцию в список Pt_CB_ACTIVATE, но добавление её в Pt_CB_ARM позволяет пользователю получить доступ к ней двумя способами:

- нажав левую кнопку мыши на виджете кнопки меню, перетащив к подсвеченному пункту меню и отпустив, чтобы это выбрать. Это известно как метод нажать-перетащить-отпустить.
 - щёлкнув на меню и затем щёлкнув на пункте меню. Если Вы используете ответную реакцию "Activate", пользователь может пользоваться только вторым методом.
4. Включите ответную реакцию, отображающую модуль меню. Смотри "Ответные реакции модуля" в главе "Редактирование ресурсов и ответных реакций в PhAB".
 5. Если Вам необходимо инициализировать меню всякий раз, когда оно отображается, задайте функцию инициализации для него. Смотри раздел "Инициализация меню" в главе "Работа с программным кодом".

Если Вы хотите, чтобы меню появлялось, когда Вы нажимаете правую кнопку мыши, в то время, когда указатель мыши установлен на виджете, Вам необходимо использовать внутреннюю связь (internal link). Для получения более полной информации см. главу "Получение доступа к модулям из программного кода" – там есть даже пример.

Модули картинок

КЛАСС ВИДЖЕТА	РАСШИРЕНИЕ ФАЙЛА	СОЗДАНИЕ ВИДЖЕТА
Не применим	.wgtp	Непосредственно из палитры виджетов

Используя модуль картинки, Вы можете применить содержание существующего модуля или создать подручную базу данных виджетов. Вы всегда отображаете картинку внутри виджета контейнерного класса или внутри другого модуля, такого как окно или диалог.



Иконка модуля картинки

Как и окна, модули картинок поддерживают множественность экземпляров. Поэтому Вы должны хранить "путь" (track) указателя экземпляра для контейнера, в котором хранится каждая картинка. При таком способе Вы будете всегда знать, с какой картинкой имеете дело, когда обрабатываете ответную реакцию. Если Вы уверены, что Ваше приложение будет использовать только один экземпляр картинки в каждый данный момент, Вам нет необходимости хранить путь указателей экземпляров. Вместо этого Вы можете использовать генерируемые PhAB декларации для получения доступа к виджетам картинки.

Отображение картинки

Вы всегда имеете доступ к модулям картинок из программного кода Вашего приложения. Чтобы получить доступ к картинке, Вы должны создать к ней внутреннюю связь. Это укажет PhAB'у генерировать декларацию, которую Вы можете использовать с функциями PhAB, такими как `ArCreateModule()`, чтобы получить доступ к картинке.

Для получения более полной информации см. главу "Получение доступа к модулям PhAB из программного кода".

Использование картинок как баз данных виджетов

Вы можете использовать модуль картинки как базу данных виджетов. База данных виджетов содержит предопределённые виджеты, которые Вы можете в любой момент скопировать в окно, диалог или контейнер.

Используя базу данных виджетов, Вам не придётся беспокоиться об обработке множественных экземпляров, поскольку сгенерированные PhAB декларации виджетов не применяются к базам данных виджетов: каждый создаваемый Вами виджет является новым экземпляром. Указатель на экземпляр возвращается Вам, когда Вы создаёте виджет, используя `ArCreateWidget()`. Если Вам понадобится в будущем получить доступ к виджету, Вам придётся вручную сохранять путь этого указателя.

Для получения более полной информации см. раздел "Базы данных виджетов" в главе "Получение доступа к модулям PhAB из программного кода".

Изменение размера модуля картинки

Не имеет значение, насколько большим или малым Вы сделали модуль картинки. Это потому, что он не ассоциирован с классом виджетов. Используются только виджеты внутри модуля.

Модули иконок

КЛАСС ВИДЖЕТА	РАСШИРЕНИЕ ФАЙЛА	СОЗДАНИЕ ВИДЖЕТА
Не применим	.wgti	Виджеты являются предопределёнными

Модули иконок позволяют Вам проектировать Ваши иконки приложения. PhAB гарантирует, что эти иконки будут автоматически поддерживаться панелью задач PhAB и плагинами запуска (launcher plugins).



Иконка модуля иконки

Модули иконок состоят из двух виджетов иконок:

- больших
- малых, предназначенных для системных панелей (shelf) и плагинов запуска.

PhAB автоматически предоставляет примитивы этих иконок, когда Вы создаёте модуль. В любой момент Вы можете использовать попиксельный редактор PhAB, описанный в главе "Редактирование ресурсов и ответных реакций в PhAB", чтобы перерисовать эти примитивы в нечто более соответствующее Вашему приложению. Вы даже можете использовать попиксельный редактор, чтобы импортировать существующую графику.

Вы можете свободно изменять размер модуля иконки по Вашему желанию – это не будет использоваться при отображении иконок.

Задание размеров и имён экземпляров

Виджеты в модуле иконки могут быть любого класса, но они должны иметь следующие размеры и имена экземпляров:

ИМЯ ЭКЗЕМПЛЯРА	РАЗМЕР
LIcon	43*43 пикселя
SIcon	15*15 пикселей

Два простых виджета иконки, обеспечиваемых PhAB, являются виджетами PtLabel с ресурсами "Label Type", установленными в Pt_IMAGE. Эти имена предварительно поименованы и их размеры предварительно заданы, чтобы соответствовать спецификации. За исключением случаев, когда у Вас есть специфические требования к иконкам, мы рекомендуем Вам использовать эти примитивы и редактировать их с помощью попиксельного редактора.

Глава 5. Создание виджетов в PhAB

Как только Вы создали или открыли приложение, Вы, вероятно, захотите добавить, удалить или модифицировать виджеты. Эта глава описывает, как работать с виджетами. Она включает:

- Типы виджетов
- Имена экземпляров
- Создание виджета
- Выбор виджетов
- Выравнивание виджетов
- Общепользовательский доступ (CUA) и обработка фокусирования
- Выстраивание виджетов
- Перетаскивание виджетов
- Установка x и y координат для виджетов
- Перемещение виджетов между контейнерами
- Изменение размеров виджетов и модулей
- Буфер обмена
- Дублирование виджетов и контейнеров
- Удаление виджетов
- Импортирование графических файлов
- Изменение классов виджетов
- Шаблоны

☞ Для получения более полной информации по использованию специфических классов виджетов, см.

- Приложение "Обзор виджетов" в этом руководстве
- "Справочник по виджетам".

Поскольку виджеты наследуют кучу свойств от своих родительских классов, Вам стоит познакомиться с фундаментальными классами: `PtWidget`, `PtBasic`, `PtContainer` и прочая.

Типы виджетов

Существуют два основных типа виджетов:

- Контейнерные виджеты, такие как `PtWindow` и `PtScrollContainer`
- Неконтейнерные виджеты, такие как `PtButton` и `PtText`.

Виджеты контейнерного класса могут содержать другие виджеты, в том числе другие контейнеры. Виджеты, расположенные внутри контейнера, известны как порождённые виджеты; иерархия, возникающая в результате такой компоновки, называется семейством виджетов. Контейнерные виджеты могут заботиться о задании размеров и позиционировании своих потомков, как описано в главе "Управление геометрией".

Работая с виджетами контейнерного класса в PhAB, помните следующее:

- если Вы перемещаете контейнер, все виджеты, порождённые контейнером, также перемещаются
- если Вы помещаете указатель внутрь контейнера, когда создаёте новый виджет, этот виджет иерархически будет размещён внутри контейнера
- если Вы хотите использовать для выбора виджетов в контейнере метод ограничивающего прямоугольника, Вы должны:
- нажать `<Alt>` перед тем как начать окаймлять
- начать окаймление внутри контейнера.

Для получения более полной информации см. раздел "Выбор виджетов" в этой главе.

Имена экземпляров

Если Ваша программа взаимодействует с виджетом, этот виджет должен иметь уникальное имя экземпляра. Используя это имя, PhAB генерирует глобальную переменную и декларацию, позволяющую Вам легко получить доступ к виджету из Вашего программного кода.

Чтобы просмотреть или отредактировать имя экземпляра виджета, используйте область "Widget Instance Name" в верхней части панели управления ресурсами или ответными реакциями:



Редактирование имени экземпляра виджета

- Имя экземпляра виджета используется для создания нескольких переменных языка C, так что оно может включать только буквы, цифры и символы подчёркивания. PhAB не позволит Вам использовать какие-либо другие символы. Имя экземпляра не может быть длиннее чем 64 символа.
- Вам имеет смысл разработать соглашение по именам для всех виджетов Вашего приложения – это сделает легче управляемыми большие приложения.

Вы можете (что необязательно) включить имя экземпляра в память виджета. См. раздел "Включение имён экземпляров" в главе "Работа с приложениями".

Принимаемое по умолчанию имя экземпляра

Когда Вы создаёте виджет, PhAB автоматически присваивает ему имя экземпляра, даваемое по умолчанию. Обычно это имя по умолчанию является именем класса виджета. Например, если Вы создаёте виджет класса PtButton, панели управления ресурсами и ответными реакциями отобразят PtButton как имя экземпляра.

Если виджет служит просто надписью или декором окна, к нему не будет выполняться доступ из программного кода приложения. Таким образом, Вы можете указать PhAB игнорировать имя экземпляра виджета во время генерирования кода. Чтобы сделать это:

- оставьте имя экземпляра эквивалентным имени класса (то есть оставьте его таким, каким принято по умолчанию)
- или
- установите пустое имя экземпляра.

Когда назначать уникальное имя

Вы можете дать виджету уникальное имя, если:

- к виджету должна быть прикреплена ответная реакция
- приложению необходимо изменять виджет установкой ресурса
- приложению необходимо извлечь из виджета информацию.

- Чтобы держать число глобальных переменных на минимуме, не присваивайте виджету уникального имени до тех пор, пока Вам действительно не понадобится получить доступ к виджету из программного кода приложения. Если Вы присвоили имя виджету и позже решили, что Вам не нужно имя, просто измените это имя обратно к имени класса виджета или сделайте это имя пустым.

Имена экземпляров и переводы

Как описано в главе "Поддержка международных языков", Вам понадобится имя экземпляра для каждой текстовой строки в интерфейсе пользователя Вашего приложения. Чтобы указать, что для генерации кода не требуется имя экземпляра, начните имя с символа @. PhAB опознает такое имя, генерируя базу данных текста языка, но пропустит его при генерировании кода. Если Вы не желаете создавать уникальное имя экземпляра для строки, которая будет переводиться, задайте одиночный символ @ в качестве имени экземпляра; PhAB добавит внутренний очередной номер в конце.

Если Вы не хотите создавать уникальное имя экземпляра, но хотите организовать текст для перевода (скажем, модулями), Вы можете присвоить строкам одно и то же имя экземпляра, и PhAB добавит к нему последовательный номер. Например, если Вы присвоили имя экземпляра "@label" нескольким строкам, PhAB сгенерирует в качестве имён экземпляров "@label", "@label0", "@label1" и т.д.

Дублированные имена

PhAB переустанавливает имя экземпляра виджета обратно в имя класса виджета, если обнаруживает дубликат имени, когда Вы:

- копируете и вставляете виджет (см. "Буфер обмена")
- импортируете виджет из другого приложения (см. раздел "Импортирование модулей PhAB из других приложений" в главе "Работа с приложениями").
- дублируете виджет (см. "Дублирование виджетов и контейнеров").

Создание виджета

Для создания виджета:

1. Щёлкните на иконке палитры виджетов на том типе виджета, который Вы хотите создать (см. приложение "Обзор виджетов" для идентификации иконок палитры виджетов).
2. Переместите указатель туда, где Вы хотите создать виджет. Указатель изменит свой вид, показывая, что делать дальше:
 - если указатель – крестик, и Вы создаёте виджет PtPolygon или PtBezier, удерживайте нажатой кнопку мыши и протащите указатель до тех пор, пока линия не протянется туда, куда Вы желаете. Чтобы добавить точки, Вы должны начать следующую линию в вершине последней. Чтобы замкнуть полигон, поместите последнюю точку в вершине первой линии.
 - если указатель – крестик и Вы создаёте виджет какого-либо другого типа, щёлкните кнопкой мыши.
 - если указатель представляет из себя стрелку с двумя наконечниками, удерживайте нажатой кнопку мыши и перетаскивайте указатель, пока виджет не станет того размера, который Вы хотите.

☞ Виджеты "цепляются" на сетке, если это включено. См. "Предустановки сетки" в главе об окружении PhAB. Чтобы улучшить производительность Вашего приложения, избегайте перекрытия виджетов, которые часто обновляются.

Вы можете также создавать виджет, перетаскивая его иконку из палитры виджетов на панель управления "Module Tree". То место, где Вы сбросили иконку, и определит место виджета в иерархии семейства.

Создание нескольких виджетов

Как только Вы создали виджет, Вы возвращаетесь в выбранный режим. Чтобы остаться в режиме создания, так чтобы Вы могли создать несколько виджетов одного типа:

1. Нажмите и удерживайте нажатой клавишу <Ctrl>.
2. Создайте столько виджетов, сколько хотите.
3. Отпустите <Ctrl>.

Отмена режима создания

Чтобы отменить режим создания, не создавая виджет:

- щёлкните где-нибудь вне модуля
или
- щёлкните на модуле правой кнопкой мыши.

Выбор виджетов

В этом разделе мы увидим, как

- выбирать одиночный виджет
- выбирать несколько виджетов
- выбирать виджеты внутри группы
- выбирать скрытые виджеты.

Когда PhAB находится в режиме выбора, указатель выглядит как стрелка. Чтобы перевести PhAB в режим выбора:

- щёлкните где-либо вне модуля
или
- щёлкните правой кнопкой мыши на модуле
или
- щёлкните на выбранном виджете в палитре виджетов.

Одиночный виджет

Чтобы выбрать одиночный виджет, Вы можете:

- указать и щёлкнуть
или
- использовать кнопки "Следующая" и "Предыдущая" в панелях управления ресурсами или ответными реакциями
- использовать панель управления "Module Tree".

Эти методы описаны ниже.

Метод указать-и-щёлкнуть

Чтобы выбрать одиночный виджет, используя указать-и-щёлкнуть:

1. Убедитесь, что Вы в режиме выбора
2. Щёлкните на виджет, используя левую кнопку мыши. Вокруг виджета появятся метки-манипуляторы изменения размеров.

Чтобы выбрать родителя данного виджета, удерживайте нажатыми <Shift>+<Alt> и щёлкните на виджет. Это лёгкий способ выбрать PtDivider или PtToolbar.

Метод контрольных панелей

Кнопки "Следующая" и "Предыдущая" на панелях управления ресурсами и ответными реакциями позволяют Вам выбрать любой виджет в текущем модуле.

ЧТОБЫ ВЫБРАТЬ	ЩЁЛКНИТЕ НА:	ИЛИ НАЖМИТЕ
Предыдущий виджет в текущем модуле	◀	<F9>
Следующий виджет в текущем модуле	▶	<F10>

Панель управления "Module Tree" отображает дерево всех виджетов в модуле. Используя это дерево, Вы можете:

- выбрать виджет внутри группы
- найти виджет по имени
- выбрать виджет, скрытый под другим виджетом.

Чтобы выбрать виджет из дерева, щёлкните на имени виджета.

Несколько виджетов

Чтобы выбрать несколько виджетов, Вы можете:

- использовать охватывающий прямоугольник
или
- использовать <Shift>+щелчок
или
- использовать панели управления.

☞ Когда Вы выбираете два или более виджета, панель управления ресурсами отображает только ресурсы, которые являются общими для этих виджетов. Редактирование какого-либо из этих ресурсов действует на все выбранные виджеты.

Использование охватывающего прямоугольника

Охватывающий прямоугольник позволяет Вам выбрать одновременно несколько виджетов:

1. Разместите указатель выше и левее виджетов, которые Вы хотите выделить.
2. Если виджеты входят в какой-то контейнер, такой как PtBkgd, убедитесь, что указатель находится внутри контейнера, затем нажмите и удерживайте клавишу <Alt>.
3. Нажмите и удерживайте левую кнопку мыши, затем протащите указатель вправо вниз. Вы увидите контур, "растущий" на экране.
4. Когда все виджеты окажутся внутри контура, отпустите кнопку мыши. Вы увидите метки-манипуляторы изменения размеров вокруг области, определённой выбранными виджетами.

Использование "<Shift> и щелчок"

Чтобы добавить или удалить виджет из текущего списка выбранных виджетов, удерживайте нажатой клавишу <Shift> и щёлкайте по виджетам. Этот метод известен также как метод расширенного выбора.

Если виджет ещё не выбран, он добавляется к списку. Если виджет уже выбран, он удаляется из списка.

☞ Вышеописанные методы выбора нескольких виджетов работают только для виджетов одного уровня иерархии. Например, Вы легко можете выбрать, скажем, две кнопки внутри окна. Вы не сможете расширять этот выбор, включив кнопку, находящуюся внутри панели (pane).

Использование панелей управления

Чтобы выбрать несколько виджетов, используйте кнопки "Следующая" и "Предыдущая" панелей управления ресурсами и ответными реакциями:

1. Удерживайте нажатой клавишу <Shift>
2. Щёлкните на кнопку "Next".

Каждый раз при щелчке PhAB добавит следующий виджет текущего модуля к выбранным.

Чтобы удалить последний виджет из текущего списка выбранных виджетов:

1. Удерживайте нажатой клавишу <Shift>
2. Щёлкните на кнопку "Previous".

Каждый раз при щелчке PhAB удалит виджет.

Виджеты внутри группы

Чтобы выбрать виджет внутри группы, Вы можете использовать кнопки "Следующая" и "Предыдущая" в панели управления ресурсами или ответными реакциями, или использовать панель управления "Module Tree".

Использование панели "Module Tree"

Чтобы выбрать один виджет внутри группы, используйте панель управления "Module Tree".

1. Переключитесь на панель управления "Module Tree".
2. Найдите группу на дереве и щёлкните на имени виджета.
3. <Shift>+щелчок – чтобы выбрать при необходимости дополнительные виджеты.
4. Чтобы отредактировать виджет, переключитесь на панель управления ресурсами или ответными реакциями.

Использование кнопок "Следующая" и "Предыдущая"

Чтобы выбрать один или более виджетов внутри группы, используйте кнопки "Следующая" или "Предыдущая":

1. Щёлкните на каком-либо виджете внутри группы, чтобы выбрать всю группу.
2. Щёлкайте на кнопке "Next" (или нажмите <F10> на панели управления ресурсами или ответными реакциями, пока не будет выбран нужный Вам виджет.
3. Чтобы выбрать дополнительные виджеты, нажмите <Shift>, затем вновь щёлкните на кнопке "Следующая".
4. Вы теперь можете редактировать ресурсы или ответные реакции виджетов.

Скрытые виджеты

Если Вы не можете найти виджет (он может быть скрыт под другим виджетом или вне границ своего контейнера), сделайте следующее:

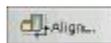
1. Используйте кнопки "Следующая" и "Предыдущая" на панелях управления ресурсами или ответными реакциями.
2. Выберите виджет из панели управления "Module Tree".
3. Используйте панель управления поиска.
4. Если виджет выглядит помещённым вне текущих границ своего контейнера, верните его обратно к видимому, используя области X и Y в панели инструментов PhAB.

Для получения более полной информации по панели инструментов и панелям управления см. главу об окружении PhAB.

Выравнивание виджетов

Вы можете выровнять несколько виджетов относительно другого виджета или их родительского контейнера.

Для простого выравнивания используйте кнопку "Align" на панели инструментов PhAB:



и затем выберите способ выравнивания во всплывающем меню.

Для более сложных вариантов выравнивания вызовите диалог "Align Widgets":

- выбрав кнопку "Align" из панели инструментов PhAB и затем выбрав "Alignment Tool" из меню
или
- выбрав "Alignment" из меню "Edit" и затем выбрав пункт "Alignment Tool" из подменю
или
- нажав <Ctrl>+<A>.

По другому виджету

Когда Вы используете этот метод для выравнивания виджетов, виджеты выравниваются по первому выбранному Вами виджету:

1. Выберите первый виджет
2. Используйте метод выбора <Shift>+щелчок, выберите остальные виджеты (этот метод описан в разделе "Выбор виджетов").
3. Для простого выравнивания выберите иконку "Align" из панели инструментов PhAB и сделайте выбор из меню.
4. Для более сложных вариантов выравнивания вызовите диалог "Align Widgets". Выберите одну или более опций выравнивания, затем щёлкните на кнопке "Align". Не щёлкайте на кнопке "Align to Container".

По родительскому контейнеру

Чтобы выровнять виджеты по их родительскому контейнеру:

1. Выберите один или несколько виджетов в любом порядке.
2. Вызовите диалог "Align Widgets", выберите желаемые опции выравнивания, затем щёлкните на соответствующей кнопке "Align to Container".

☞ Если Вы выбрали и вертикальную, и горизонтальную опции, убедитесь, что щёлкнули на обе кнопки "Align to Container".

3. Щёлкните на кнопке "Align".

При выравнивании виджетов по контейнеру Вам может понадобиться сохранить их относительные позиции. Чтобы сделать это:

1. Сгруппируйте виджеты вместе (см. раздел "Выравнивание виджетов, используя группы") в главе "Управление геометрией".
2. Выровняйте виджеты.
3. Разбейте, что необязательно, группу на составляющие.

Общепользовательский доступ (СИА) и управление фокусом

Общепользовательский доступ (СИА) является стандартом, описывающим, как пользователь может изменить фокус с помощью клавиатуры. Виджет является фокусируемым, если на него может устанавливаться фокус путём нажатия клавиши СИА или вызовом функции фокусировки.

Изменение фокусировки через клавиатуру

Следующие клавиши перемещают фокус только к фокусируемым виджетам:

ЧТОБЫ ПЕРЕЙТИ К:	НАЖМИТЕ
следующему виджету	Tab
предыдущему виджету	Shift+Tab
первому виджету в следующем контейнере	Ctrl+Tab
последнему виджету в предыдущем контейнере	Ctrl+Shift+Tab

Чтобы получить информацию о задании порядка, в котором виджеты отслеживаются, см. раздел "Упорядочивание виджетов" в этой главе.

Управление фокусом

Для управления фокусом для виджета используйте следующие флаги:

Pt_ARG_FLAGS:

Pt_GETS_FOCUS

Сделать виджет фокусируемым

Pt_FOCUS_RENDER

Сделать виджет подающим визуальную информацию, когда он имеет фокус.

Дополнительно, используйте для управления фокуса для контейнера следующие флаги

Pt_ARG_CONTAINER_FLAGS:

Pt_BLOCK_CUA_FOCUS

Предотвращает использование клавиши CUA для входа в контейнер. Однако, если пользователь щёлкнет внутри контейнера, или функция фокусирования даст контейнеру фокус, клавиши CUA могут использоваться.

Pt_ENABLE_CUA

Даёт родительскому виджету возможность задавать, обрабатывает или не обрабатывает порождённый контейнер клавиши CUA:

- если этот флаг установлен, код виджета обрабатывает клавиши CUA
- если он не установлен, клавиши CUA пропускают семейство виджета, пока не будет найден прародитель с установленным флагом. Этот прародитель (если он найден) обрабатывает клавиши.

Pt_ENABLE_CUA_ARROWS

Аналогичен флагу Pt_ENABLE_CUA, но используется только клавишами стрелок.

Ответные реакции фокусировки

Все потомки виджета PtBasic имеют следующие ресурсы ответных реакций:

- Pt_CB_GOT_FOCUS – вызывается, когда виджет получает фокус
- Pt_CB_LOST_FOCUS – вызывается, когда виджет теряет фокус.

Виджет может даже отказаться "отдавать" фокус (например, если Вы набрали ошибочную дату в текстовом виджете).

☞ PtMultiText и PtText имеют специальные версии этих ответных реакций.

Для получения более полной информации см. "Справочник виджетов".

Функции обработки фокусировки

Функции, описанные ниже, имеют дело с фокусировкой. Они описаны в "Справочнике библиотечных функций Photon".

В действительности эти функции не изменяют виджеты, имеющие фокус; они сообщают Вам, где фокус может проходить:

PtFindFocusChild()	Отыскивает ближайший фокусируемый порождённый виджет.
PtFindFocusNextFrom()	Отыскивает следующий виджет, который может получить фокус.
PtFindFocusPrevFrom()	Отыскивает предыдущий виджет, который может получить фокус.

Вы можете использовать эти функции, чтобы определить, какой виджет имеет фокус:

PtContainerFindFocus()	Отыскивает виджет, имеющий фокус, в той же семейной иерархии, что и виджет.
PtIsFocused()	Определяет, на каком уровне виджет имеет фокус.

Вы можете использовать эти функции, чтобы передать фокус виджету:

PtContainerFocusNext()	Передаёт фокус следующему Pt_GETS_FOCUS виджету.
PtContainerFocusPrev()	Передаёт фокус предыдущему Pt_GETS_FOCUS виджету.
PtContainerGiveFocus() или PtGiveFocus()	Передаёт фокус виджету – эти функции идентичны.
PtContainerNullFocus()	Отменяет фокус для виджета.
PtGlobalFocusNext()	Передаёт фокус следующему виджету.
PtGlobalFocusNextContainer()	Передаёт фокус виджету следующего контейнера.
PtGlobalFocusNextFrom()	Передаёт фокус виджету, следующему за заданным.
PtGlobalFocusPrev()	Передаёт фокус предыдущему виджету.
PtGlobalFocusPrevContainer()	Передаёт фокус виджету предыдущего контейнера.
PtGlobalFocusPrevFrom()	Передаёт фокус виджету, находящемуся перед заданным.

Упорядочивание виджетов

В PhAB каждый виджет присутствует перед или позади других виджетов. Это известно как порядок виджетов, и Вы можете видеть его, когда несколько виджетов перекрываются. Порядок виджетов диктует то, как Вы можете использовать клавиши CUA для перемещений между виджетами.

☞ Если Вы не использовали PhAB, порядок виджетов – это порядок, в котором виджеты были созданы. Чтобы изменить порядок, см. раздел "Упорядочивание виджетов" в главе "Управление виджетами в программном коде приложения".

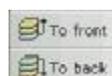
Чтобы просмотреть порядок виджетов, сделайте одно из следующего:

- используйте панель управления "Module Tree". Виджеты для каждого контейнера приведены в списке от заднего к переднему;
или
- используйте режим тестирования и нажмите клавишу <Tab>, повторяя это, чтобы проверить порядок фокусировки.

Простейший путь переупорядочить виджеты – это использовать панель управления "Module Tree": просто перетаскивайте виджеты, пока не установите их в требуемом порядке.

Для переупорядочивания виджетов Вы можете также использовать метод "Shift-выбора":

1. Используя расширенный метод выбора ("<Shift> и щелчок"), выберите виджеты в требуемом Вами порядке (этот метод выбора описан в разделе "Выбор виджетов")
2. Сделайте одно из следующего:
 - выберите "To Front" или "To Back" из меню "Edit"
 - нажмите <Ctrl>+<F> или <Ctrl>+.
 - щёлкните на одной из этих иконок:



PhAB разместит виджеты в порядке, в котором Вы их выбрали.

Вы можете также выбрать один или более виджетов и затем использовать иконки "Raise" и "Lower", чтобы изменить порядок виджетов:



Перетаскивание виджетов

В большинстве ситуаций перетаскивание виджета – это самый лёгкий способ переместить виджет, поскольку это выполняется быстро и довольно точно:

1. Выберите виджет.
2. Укажите на один из выбранных виджетов, нажмите кнопку мыши и, удерживая её, перетащите виджеты в новое положение.

Если Вы хотите перетаскивать виджеты горизонтально, вертикально или по диагонали, удерживайте нажатой при перетаскивании клавишу <Alt>.

☞ Чтобы перетаскивать контейнер горизонтально, вертикально или по диагонали, нажмите клавишу <Alt> после того, как нажали кнопку мыши (нажатие <Alt> перед нажатием кнопки мыши выберет виджеты внутри контейнера).

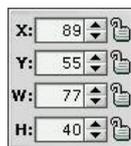
3. Отпустите кнопку мыши. Виджет "зацепится" за сетку, если это дозволено – см. раздел "Предустановки сетки" в главе об окружении PhAB.

Чтобы отменить операцию перетаскивания, нажмите клавишу <Esc> перед тем, как отпустить кнопку мыши. Чтобы переместить родительский контейнер виджета, удерживайте нажатыми <Shift>+<Alt> и перетаскивайте виджет.

Если Вы перемещаете виджеты за пределы их контейнеров, они могут "исчезать". Если такое приключилось, используйте кнопки "Предыдущая" и "Следующая" на панели управления ресурсами или ответными реакциями, чтобы выбрать виджеты, затем используйте области "X" и "Y" на панели инструментов PhAB, чтобы вернуть виджеты обратно в область видимости.

Если Вы обнаружите, что нечаянно перетаскиваете виджеты, тогда как хотите их просто выбрать, Вам следует принять следующие меры:

- установите фактор демпфирования, как описано в разделе "Предустановки перетаскивания" ниже
- запретите координаты виджета и/или размеры его в панели инструментов PhAB



Для получения более полной информации см. раздел "Панели инструментов" в главе об окружении PhAB.

Предустановки перетаскивания

Существует несколько предустановок, которые Вы можете установить для перетаскивания (см. раздел "Подгонка Вашего окружения PhAB" в главе об окружении PhAB):

- перетаскивание имеет фактор демпфирования, определяющий, как далеко Вы должны перетащить виджет до того, как он переместится. По умолчанию он равен 4 пикселям.
- Вы можете перетаскивать виджеты либо как контуры, либо как полные виджеты.
- Вы можете перетаскивать модули либо как контуры, либо как полные модули.

Установка координат x и y виджетов

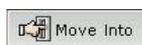
Чтобы разместить один или более виджетов в заданных координатах:

1. Выберите виджеты.
2. Введите координаты в областях x и y панели инструментов PhAB, затем нажмите <Enter>. Для получения более полной информации см. главу об окружении PhAB.

Перемещение виджетов между контейнерами

Чтобы переместить один или более виджетов непосредственно из одного контейнера или модуля в другой:

1. Выберите виджеты.
2. Выполните одно из следующего:
 - выберите "Transfer" из меню "Edit"
или
 - нажмите <Ctrl>+<T>
или
 - щёлкните на иконке "Move Into" на панели инструментов PhAB:



3. Переместите указатель на другой контейнер и щёлкните на кнопке мыши.

Изменение размеров виджетов и модулей

Когда Вы выбираете виджет или модуль, Вы видите его высоту и ширину – включая все границы и края – отображёнными в полях H и W панели инструментов (эти значения содержатся в ресурсе Pt_APG_DIM; см. описание PtWidget в "Справочнике виджетов"). Чтобы изменить размеры выбранного виджета, выполните одно из следующего:

- перетащите метку-манипулятор изменения размеров виджета
или
- щёлкните на областях высоты или ширины панели инструментов PhAB; наберите новое значение, затем нажмите <Enter>. Для получения более полной информации см. главу об окружении PhAB
или
- используйте инструмент "подталкивание" панели инструментов.

☞ Если модуль в режиме "Test", Вы не сможете изменить размеры его виджетов. Если у Вас затруднения в том, чтобы увидеть метки-манипуляторы изменения размеров, поскольку они сливаются с цветом фона, Вы можете изменить цвет этих меток. Для получения более полной информации см. раздел "Подгонка под себя окружения PhAB" в главе "Окружение PhAB".

Буфер обмена

Буфер обмена PhAB позволяет ВАМ вырезать, копировать и вклеивать виджеты. Вы не можете использовать этот буфер с другими приложениями, но можете использовать для копирования или перемещения виджетов с одного приложения PhAB в другой.

Вы можете найти буфер обмена полезными по следующим соображениям:

- он оберегает Вас от необходимости создания большого количества виджетов из одного "полуфабриката"
- он помогает Вам создавать приложения, в которых виджеты выглядят и ведут себя согласованно друг с другом.

Вырезание и копирование

Операция вырезания удаляет выбранные виджеты из своего модуля и размещает их в в буфере обмена. Операция копирования копирует выбранные виджеты в буфер обмена, не удаляя их из модуля.

☞ И при вырезании, и при копировании PhAB удаляет любые виджеты, которые уже были в буфере обмена.

Чтобы вырезать или скопировать один или более виджетов:

1. Выберите виджеты.
2. Чтобы вырезать, выполните одно из следующего:
 - выберите пункт "Cut" из меню "Edit"
или
 - нажмите
или
 - щёлкните на иконке "Cut" на панели инструментов PhAB:



3. Чтобы скопировать, выполните одно из следующего:
 - выберите пункт "Copy" из меню "Edit"
или
 - нажмите <Shift>+
или
 - щёлкните на иконке "Copy" на панели инструментов PhAB:



- ☞
- Когда Вы вырезаете или копируете виджет в буфер обмена, PhAB удаляет его ответные реакции. Если Вы хотите переместить виджет в другой контейнер, но сохранить его ответные реакции, см. "Перемещение виджетов между контейнерами" в этой главе
 - Меню "Edit" содержит также команду "Delete". Эта команда навсегда удаляет виджеты без копирования их в буфер обмена.

Вклеивание

Операция вклеивания копирует виджеты из буфера обмена в модуль. Чтобы вклеить содержимое в буфер обмена:

1. Убедитесь, что Вы в режиме выбора.
2. Выполните одно из следующего:
 - выберите команду "Paste" из меню "Edit"
или
 - нажмите <Insert>
или

- щёлкните на иконке "Paste" на панели инструментов PhAB:



3. Укажите, где Вы хотите, чтобы объекты буфера обмена появились, затем щёлкните мышью.

- Имена экземпляров обычно при вклеивании сохраняются. Но если PhAB обнаруживает дублирование имени, он изменяет это имя обратно к имени класса.
- Поскольку состояние буфера обмена сохраняется между приложениями PhAB, Вы можете вырезать виджеты из одного приложения PhAB и вклеивать их в другое.

Просмотр буфера обмена

Чтобы просмотреть содержимое буфера обмена, выберите пункт "Clipboard" из меню "View".

Редактирование буфера обмена

Чтобы добавить новые виджеты в буфер обмена, или чтобы редактировать или удалять конкретные виджеты, уже находящиеся в буфере обмена, используйте те же методы, которые Вы использовали бы при редактировании виджетов в окне или модуле диалога.

Более быстрый способ дублировать виджеты описаны в разделе "Дублирование виджетов и контейнеров". Чтобы изучить более быстрый способ перемещать виджеты с одного контейнера в другой см. раздел "Перемещение виджетов между контейнерами" в этой главе.

Дублирование виджетов и контейнеров

Вот быстрый и простой способ дублировать виджеты и контейнеры (он проще чем использование буфера обмена):

1. Нажмите и удерживайте нажатой клавишу <Ctrl>.
 2. Укажите на виджет или контейнер, нажмите и удерживайте левую клавишу мыши, затем перетащите указатель мыши туда, где Вы желаете, чтобы новый виджет появился.
 3. Отпустите клавишу <Ctrl> и кнопку мыши.
- Одновременно Вы можете дублировать только один контейнер или виджет. Если Вы дублируете виджет, дублируются все его порождения.
 - Имена экземпляров новых виджетов переустанавливаются в имя класса виджета.
 - Ответные реакции не дублируются.

Удаление виджетов

Для окончательного удаления одного или нескольких выбранных виджетов без копирования их в буфер обмена:

1. Выберите виджеты.
2. Выберите команду "Delete" из меню "Edit" или нажмите <Ctrl>+<D>.

Если Вы хотите поместить виджеты где-то в другом месте, используйте вырезание виджетов, а не их удаление. Для получения более информации см. раздел "Буфер обмена" в этой главе.

Импортрование графических файлов

PhAB позволяет Вам импортровать в приложение несколько разновидностей графических файлов. Для получения более полной информации см. раздел "Импортрование файлов" в главе "Работа с приложениями".

☞ PhAB не выполняет экспортрование графических файлов напрямую, потому что любые импортруемые файлы сохраняются в модуле в специфическом формате PhAB.

Попиксельный редактор (описанный в главе "Редактирование ресурсов и ответных реакций в PhAB) также позволяет Вам импортровать графику: выберите виджет, который Вы хотите добавить в образ, отредактируйте его вид и выберите кнопку "Import" попиксельного редактора.

Изменение класса виджета

Вы можете изменить класс виджета, выбрав его и затем выбрав команду "Change Class" из меню "Edit". Выберите новый класс из всплывшего списка и затем щёлкните на кнопке "Change class".

Ресурсы и ответные реакции, которые совместимы с новым классом виджета, сохраняются вместе со своими значениями. Например, если Вы решили, что PtMultitext лучше согласуется с Вашими нуждами, чем PtButton, Вы можете выбрать кнопку, открыть диалог "Change Class", щёлкнув правой кнопкой на виджете, или щёлкнув правой кнопкой в дереве модулей, или выбрав команду "Change Class" из меню "Edit". Позиция виджета, размеры, Pt_ARG_TEXT_STRING и все другие ресурсы, общие для старого и нового класса, сохраняются.

☞ Когда Вы изменяете класс виджета, некоторые ресурсы и ответные реакции могут быть удалены. Перед исполнением диалог отобразит число ресурсов и ответных реакций, которые будут удалены. У Вас есть шанс отменить операцию.

Контейнер, имеющий порождения (такой как PtPanel с несколькими виджетами внутри него) не может быть конвертирован подобным образом; пункт меню "Change Class" в этом случае не включается.

Шаблоны

Шаблон – это "подогнанный под себя" виджет, группа или иерархия виджетов, которые Вы бы хотели использовать как основу для других виджетов. Шаблоны полезны, когда Вы хотите создавать много виджетов, которые выглядят и ведут себя сходным образом. PhAB автоматически загружает Ваши шаблоны, так что Вы можете легко создать экземпляры Ваших виджетов в любом приложении.

☞ Чтобы посмотреть примеры создания шаблона, см. раздел "Редактирование ресурсов" в главе "Уроки".

Этот раздел включает:

- Создание шаблонов
- Добавление подогнанного виджета
- Редактирование шаблонов
- Удаление шаблонов

Создание шаблонов

Чтобы создать шаблон:

1. Создайте и отредактируйте виджет или виджеты, как Вам требуется.
2. после выбора виджета (-ов), выберите пункт "Save as template" из меню "Edit" или из меню, появляющегося при щёлкании правой кнопкой мыши на панели управления "Module Tree".
3. Появится диалог "Save template".

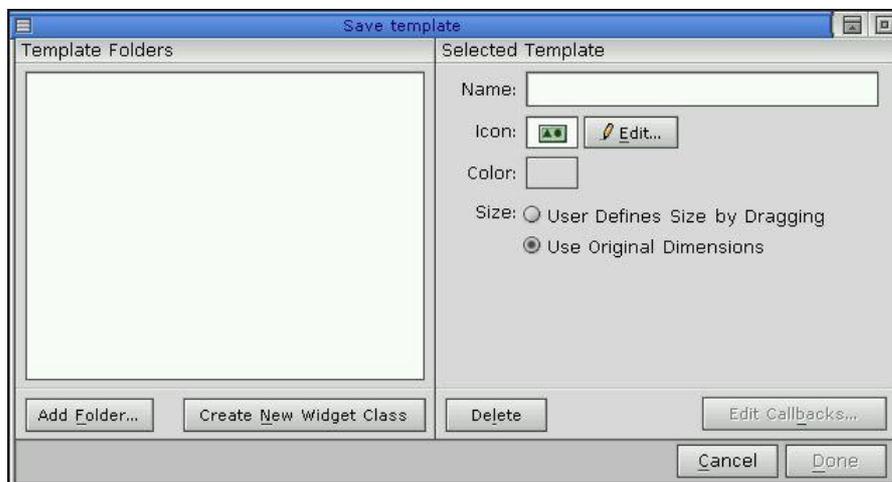


Рис. 5-1. Диалог для создания новых шаблонов

4. Выберите папку, в которой разместите новый шаблон. Чтобы заменить существующий шаблон, выберите вместо папки шаблон. Чтобы создать новую папку, щёлкните на кнопке "Add Folder" и введите имя папки. Каждая папка всплывает как палитра рядом с палитрой виджетов. Вы можете закрыть их и позднее просмотреть их, используя меню "View"; внизу этого меню находится список со всеми описанными папками. Когда Вы запускаете PhAB, по умолчанию всплывают все палитры.
5. Вы должны обеспечить шаблону имя и иконку.

☞ Диалог "Save template" не отображает иконку, которую Вы создаёте для шаблона.

6. Вы можете, что необязательно, установить цвет фона для иконки на палитре виджетов, и метод изменения размеров (использовать первоначальные размеры или изменять перетаскиванием).
7. Если виджеты, сохраняемые Вами как шаблоны, имеют прикрепленные ответные реакции, Вы можете щёлкнуть на кнопке "Edit Callbacks" и установить ответные реакции, сохраняемые вместе с шаблоном. По умолчанию сохраняются все ответные реакции.

Когда Вы щёлкните на вновь созданном шаблоне и инициализируете виджет как новый экземпляр, выбранные ответные реакции будут созданы и добавлены автоматически. Диалог отобразит список ответных реакций и Вы сможете выбрать те, которые Вы хотите добавить.

Добавление подогнанного виджета

Если Вы создали подогнанный виджет (см. "Построение подогнанных виджетов") и хотите создать из него шаблон, выберите любой одиночный виджет в Вашем приложении, откройте диалог "Save as template", следуйте шагам, приведенным выше, и щёлкните на кнопке "Create New Widget Class".

☞ Эта операция не изменит никоим образом выбранный виджет.

Диалог позволяет Вам ввести имя класса (напр., MyWidget), который будет сохранён в шаблоне. Для того, чтобы подогнанный виджет был функциональным, добавьте раздел "MyWidget" в один из файлов палитры *.pal.

- ☞ Чтобы избежать путаницы, не начинайте имя подогнанного виджета с Pt – оставьте этот префикс для стандартных виджетов Photon'a.

Редактирование шаблонов

Чтобы редактировать существующий шаблон:

1. Выберите пункт "Edit Templates" из меню "File".
2. Отобразится диалог, похожий на тот, что используется при создании шаблона.
3. Редактируйте шаблон, как Вам требуется, и затем сохраните результаты.

Удаление шаблонов

Чтобы удалить шаблон:

1. Выберите пункт "Редактирование шаблонов" из меню "Edit".
2. Отобразится диалог, похожий на тот, что используется при создании шаблона.
3. Выберите папку шаблона и щёлкните на "Delete".

Глава 6. Редактирование ресурсов и ответных реакций в PhAB

Как только Вы создали в PhAB виджеты, Вам, вероятно, понадобится редактировать их ресурсы и ответные реакции. В этой главе обсуждается:

- Редактирование ресурсов виджета
- Попиксельный редактор
- Редактор цветов
- Редактор флагов/опций
- Редактор шрифтов
- Редактор списков
- Редактор чисел
- Текстовые редакторы
- Редактор функций
- Ответные реакции
- Модульные ответные реакции
- Программные ответные реакции
- Ответные реакции горячих клавиш
- Обработчики событий – необработанные и фильтрующие

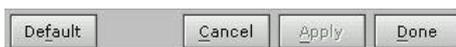
Редактирование ресурсов виджета

Обычно виджет имеет множество ресурсов, позволяющих Вам изменить его вид и поведение. Каждый тип ресурса имеет свой собственный редактор.

Чтобы открыть какой-либо редактор ресурсов:

1. Выберите один или более виджетов.
 - ☞ Когда Вы выбираете два или более виджета, панель управления ресурсами отображает только те ресурсы, которые являются общими для этих виджетов. Редактирование какого-либо из этих ресурсов действует на все выбранные виджеты.
2. Переключитесь, если необходимо, на панель управления ресурсами. Если значение ресурса было изменено с принимаемого в PhAB по умолчанию, маркировка ресурса отображается жирным шрифтом.
 - ☞ Значение ресурса, принимаемое в PhAB по умолчанию, не обязательно является значением, принимаемым самим виджетом.
3. Щёлкните на ресурсе в панели управления. Всплывёт соответствующий редактор ресурса.

Каждый редактор ресурса предоставляет следующие кнопки:



Кнопки, общие для редакторов ресурсов

КОГДА ВЫ ХОТИТЕ:	ИСПОЛЬЗУЙТЕ КНОПКУ:
Восстановить значение ресурса, принимаемое по умолчанию (или значения, если выбраны более одного виджета)	Default
Отменить все изменения, выполненные с того момента, как Вы открыли редактор или последний раз щёлкнули на кнопке "Apply"	Cancel
Применить все изменения и продолжить редактирование	Apply
Применить все изменения и закрыть редактор	Done

Редакторы для различных типов ресурсов описаны в следующих секциях:

ЧТОБЫ РЕДАКТИРОВАТЬ:	СМ. РАЗДЕЛ:
Образы	Попиксельный редактор
Цвета	Редактор цветов
Флаги	Редактор флагов/опций
Шрифты	Редактор шрифтов
Списки текстовых пунктов	Редактор списков
Чисел	Редактор чисел
Одно- или многострочные текстовые строки	Текстовые редакторы
Функции	Редактор функций

Попиксельный редактор

Попиксельный редактор позволяет Вам переделать попиксельный образ виджета. Редактор предоставляет широкий диапазон инструментов, так что Вы можете виртуально нарисовать любой образ Вашего приложения, какой Вам потребуется.

В этом разделе мы обсудим:

- установку размеров попиксельного образа
- выбор цветов
- как рисовать и вытирать
- рисование от руки
- рисование линий, прямоугольников и кругов
- заполнение замкнутых областей
- выбор области
- использование панели инструментов
- другие средства управления редактора

☞ Чтобы открыть попиксельный редактор для любого виджета, который может содержать образ (напр., PtLabel, PtButton), щёлкните на виджете, затем щёлкните на "Image resource" в панели управления ресурсами.

☞

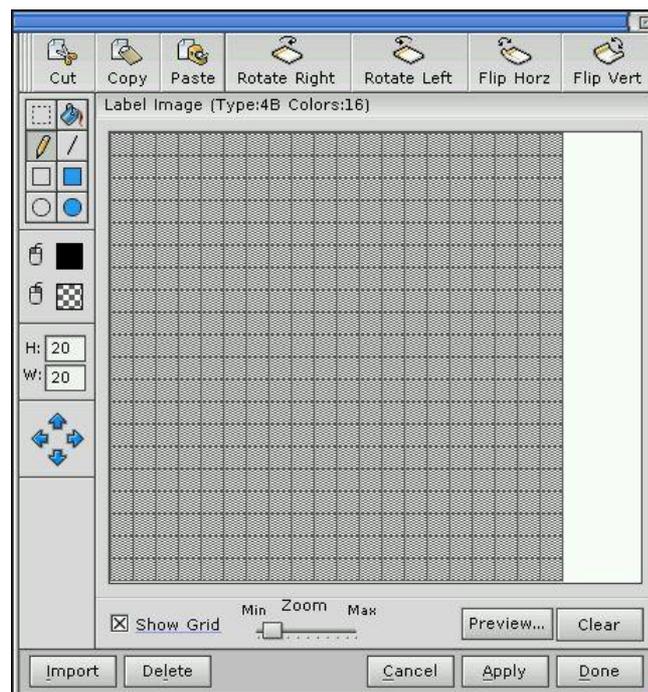


Рис. 6-1. Простая сессия попиксельного редактора

Редактор имеет несколько режимов и инструментов рисования, описанных в нижеследующих разделах. По умолчанию принят режим рисования "от руки" – Вы просто водите указателем по сетке рисования.

Установка размеров попиксельного образа

Редактор содержит области высоты и ширины образа, оба задаются в пикселях. Чтобы изменить размеры, отредактируйте значение в области и нажмите <Enter>.

☞ Если Вы уменьшили размер образа, часть рисунка может быть обрезана.

Как рисовать и стирать

Следующее относится ко всем инструментам рисования:

ДЛЯ ТОГО, ЧТОБЫ:	ИСПОЛЬЗУЙТЕ
Рисовать текущим цветом	Левую кнопку мыши
Удалить пиксель или область (т.е. рисовать цветом фона)	Правую кнопку мыши

Выбор цветов

Чтобы выбрать цвет рисования:

1. Щёлкните на следующий селектор цвета:



2. Отобразится палитра селектора цвета. Щёлкните на выбранном цвете. Всё будет рисоваться этим цветом до тех пор, пока Вы не выберете новый цвет.

Выбор фонового цвета

Фоновый цвет (или стирание) используется, когда Вы рисуете правой кнопкой мыши. Чтобы выбрать цвет фона:

1. Щёлкните на следующем селекторе цвета:



2. Щёлкните на выбранном Вами цвете.

Для получения более полной информации см. раздел "Редактор цвета".

Рисование "от руки"

Инструмент рисования "от руки" позволяет Вам рисовать линии свободной формы и удалять единичные пиксели для быстрого исправления помарок.

Чтобы рисовать в режиме "от руки":

1. Щёлкните на инструменте рисования "от руки":



2. Установите указатель в точку начала рисования.
3. Протаскивайте указатель, двигая его так, как будто Вы рисуете карандашом, затем, когда Вы выполните задуманное, отпустите кнопку мыши.
Вы можете повторять этот шаг столько раз, сколько Вам вздумается.

☞ Чтобы стереть пиксель под указателем, щёлкните правой кнопкой мыши.

Рисование линий, прямоугольников и кругов

Чтобы рисовать линии, прямоугольники или круги, используйте один стандартный метод:

1. Щёлкните на соответствующем инструменте.
2. Укажите, где Вы хотите начать рисовать объект.
3. Протяните указатель туда, где объект должен закончиться, затем отпустите кнопку мыши. Вы можете повторять этот шаг столько раз, сколько Вам вздумается.

Заполнение замкнутой области

Чтобы заполнить какую-либо замкнутую область (т.е. любую область сделать одного цвета):

1. Щёлкните на инструменте заполнения:



2. Переместите указатель внутрь области, которую Вы хотите заполнить, затем щёлкните.

☞ Если линия, ограничивающая область, разорвана, операция заполнения приведёт к выливаю краски через дырку и может привести к заполнению цветом всего попиксельного рисунка.

Выбор области

Для того, чтобы использовать некоторые инструменты, Вам необходимо вначале выбрать область на рисунке.

Чтобы выбрать область:

1. Щёлкните на инструменте выбора:



2. Укажите, где Вы хотите начать выбор.

3. Перетаскивайте указатель туда, где Вы желаете завершить выбор, затем отпустите кнопку мыши.

Вы теперь можете слегка передвинуть ("подтолкнуть") область или выполнить любое действие, описанное ниже в разделе "Использование панели инструментов попиксельного образа".

"Подталкивание" области

Чтобы "подтолкнуть" выбранную область на один пиксель:

1. Выберите область, которую хотите "подтолкнуть".
2. Щёлкнуть на стрелке подталкивания:

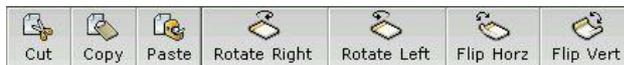


Следует отметить несколько вещей:

- PhAB переписывает все пиксели в направлении подталкивания
- Подталкивание не может вытолкнуть область за пределы рисунка
- PhAB разместит пустую область на рисунке, чтобы заполнить место, оставшееся после того, как область передвинута.

Использование панели инструментов попиксельного образа

Попиксельный редактор обеспечивает использование нескольких других инструментов из своей панели инструментов:



Вы можете выбрать область и затем использовать эти инструменты для вращения, зеркального отображения, вырезания, копирования или вклеивания этой области.

Несколько замечаний:

- Если Вы вращаете область, которая не является чисто квадратной, область может перерисовать некоторые символы.
- Если часть вращаемой области вышла за рисунок, эта часть может быть удалена.
- Используя инструмент зеркального отображения совместно с инструментом копирования, Вы можете создавать зеркальные образы.
- При вклеивании укажите на новое местоположение, затем щёлкните. Эта позиция является левым верхним углом вклеиваемой области.
- Вы можете использовать буфер обмена для копирования образа с одного виджета в другой, или копировать образ в его "установочную" версию, чтобы сделать минимальные изменения.

Другие средства управления попиксельным образом

Попиксельный редактор также включает следующие кнопки:

КОГДА ВЫ ХОТИТЕ:	ИСПОЛЬЗУЙТЕ ЭТО СРЕДСТВО:
Включить/выключить сетку	Show Grid
Просматривать меньшую или большую область рисунка	Zoom
Быстро удалить рисунок	Clear
Просмотреть рисунок в его действительных размерах	Preview
Удалить рисунок и выключить попиксельный редактор	Delete
Импортировать образ (только для ресурсов типа образов)	Import

Описание стандартных кнопок редактора в нижней части редактора см. в разделе "Редактирование ресурсов виджета".

Редактор цвета

Редактор цвета позволяет Вам модифицировать любой ресурс цвета. Там, где Вы щёлкните на панели управления ресурсами, определит, какой редактор цвета Вы увидите:

ЧТОБЫ ИСПОЛЬЗОВАТЬ:	ЩЁЛКНИТЕ НА РЕСУРСЕ ЦВЕТА:
Полный редактор цвета	Имя или описание
Быстрый редактор цвета	Текущее значение

Полный редактор цвета

Если Вы щёлкните на имени ресурса или описании (т.е. левой стороне панели управления ресурсами), отобразится полный редактор цвета:

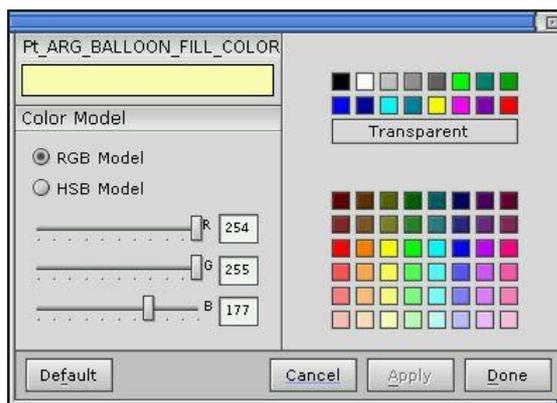


Рис. 6-2. Редактор цвета

Полный редактор цвета даёт Вам выбор из:

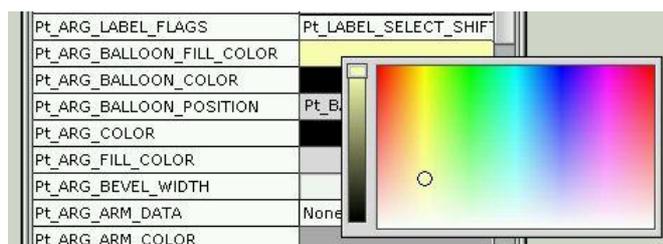
- 16 базовых цветов, которые Вы не можете изменить
- прозрачности
- 48 дополнительных цветов, которые Вы можете подогнать под свои нужды, используя ползунки либо в RGB (красеый/зелёный/синий), либо в HSB (оттенок/насыщенность/яркость) режимах цветности.

Использование прозрачного заполнения может привести к мельканию и ухудшению производительности Вашего приложения, особенно, если Ваше приложение модифицирует много объектов.

Описание стандартных кнопок редактора в нижней его части см. в разделе "Редактирование ресурсов виджета".

Быстрый редактор цвета

Если Вы щёлкните на значении ресурса цвета (т.е. на правой стороне панели управления ресурсами), отобразится быстрый редактор цвета:



Чтобы изменить значение, перемещайте ползунок слева. Для изменения оттенка щёлкните и перетаскивайте указатель на цветном лоскутке справа.

Редактор флагов/опций

Всякий раз, когда Вы щёлкаете на ресурсе флагов или ресурсе, позволяющем Вам выбирать только одно из нескольких predetermined значений, Вы увидите редактор флагов/опций. Например:

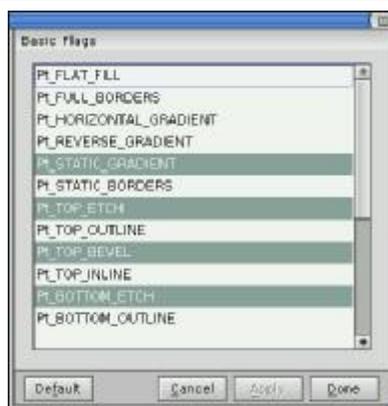


Рис. 6-3. Редактор флагов/опций

Ресурсы флагов

Если Вы щёлкните на ресурсе флагов, этот редактор позволит Вам выполнить множественные выборы из отображаемого списка. Чтобы редактировать список флагов:

1. Выберите (или снимите выбор) с желаемых флагов. Поскольку каждый флаг является отдельным переключателем, Вы можете выбрать любое число флагов или оставить их все невыбранными.
2. Примите эти изменения кнопкой ("Apply" или "Done").
Виджеты изменятся, отражая новые флаги.

Ресурсы списка опций

Если Вы щёлкните на ресурсе, который может иметь только одно значение, редактор флагов/опций позволит Вам выполнить только один выбор из отображаемого списка. Чтобы выбрать опцию из списка:

1. Щёлкните на опции. Ранее выбранная опция станет невыбранной.
2. Дайте команду принять сделанные изменения.

Описание стандартных кнопок редактора в нижней его части см. в разделе "Редактирование ресурсов виджета".

Редактор шрифтов

Каждый раз, когда Вы выберете какой-либо ресурс шрифта в панели управления ресурсами, Вы увидите редактор шрифтов:



Рис. 6-4. Редактор шрифтов

Редактор шрифтов включает такие виджеты:

Font	Гарнитура виджета. Просто щёлкните на этой кнопке или расположенной рядом кнопке меню, затем выберите из отобразившегося списка гарнитур.
Style	Стиль (жирный и/или наклонный), если возможно для текущего шрифта и его размера.
Quality	Плоский или сглаженный. Сглаживание сглаживает края у шрифта, делая шрифт выглядящим более приятно, это возможно только для масштабируемых шрифтов, таких как Swiss или Dutch.
Size	Размер шрифта в пунктах

☞ Если гарнитура не поддерживает жирный или наклонный стиль для данного типоразмера, соответствующие стили становятся тусклыми или невыбираемыми.

Описание стандартных кнопок редактора в нижней его части см. в разделе "Редактирование ресурсов виджета".

Редактор списка

Виджеты, такие как PtList, обеспечивают список базирующихся на тексте пунктов. Для редактирования списка используется редактор списка PhAB.

Чтобы открыть редактор и добавить первый пункт:

1. Выберите виджет, затем щёлкните на соответствующем ресурсе в панели управления ресурсами (обычно "List of Items"). Вы увидите редактор:

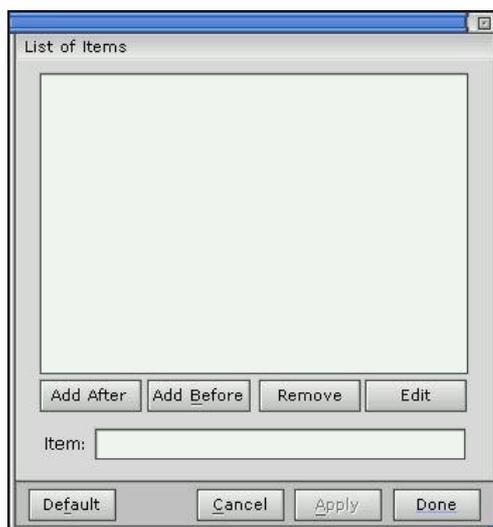


Рис. 6-5. Редактор списка

2. Щёлкните на текстовой области внизу диалога, затем наберите требуемый текст.

☞ Если Вам надо набрать символы, которых нет на Вашей клавиатуре, Вы можете использовать формирующие последовательности, приведённые в разделе "Формирующие последовательности Photon'a" в приложении "Многоязыковая поддержка Unicode".

3. Нажмите <Enter> или щёлкните на "Insert After".
4. Щёлкните на "Apply" или "Done".

Чтобы добавить ещё пункты в список:

1. Щёлкните по имеющемуся пункту, затем щёлкните на "Insert After" или "Insert Before". Вы увидите новый пункт, добавленный в список.
2. Используя текстовую область, отредактируйте текст нового пункта.

3. Щёлкните на "Modify", затем щёлкните на "Apply" или "Done".

☞ Вы не сможете создать пустые строки внутри списка пунктов.

Редактирование существующих пунктов списка

Чтобы редактировать пункт:

1. Щёлкните на пункте.
2. Отредактируйте текст пункта.
3. Щёлкните на "Modify", затем щёлкните на "Apply" или "Done".

☞ Относительно горячих клавиш текстового редактирования см. раздел "Текстовые редакторы".

Удаление пунктов списка

Чтобы удалить пункт:

1. Щёлкните на пункте, затем щёлкните на "Delete".
2. Щёлкните на "Apply" или "Done".

Описание стандартных кнопок редактора в его нижней части см. в разделе "Редактирование ресурсов виджета".

Редактор чисел

Вы можете редактировать значение числового ресурса непосредственно в панели управления ресурсами, или же Вы можете щёлкнуть на имени ресурса, чтобы использовать редактор чисел:

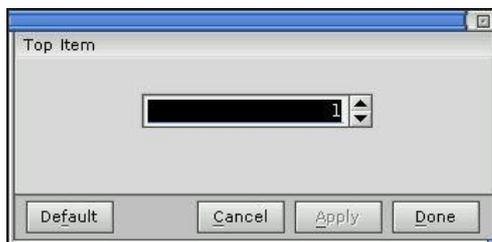


Рис. 6-6. Редактор чисел

Чтобы изменить значение, отображаемое редактором, Вы можете:

- использовать технологию редактирования текста, описанную в разделе "Текстовые редакторы" или
- щёлкнуть на кнопках увеличения/уменьшения.

Описание стандартных кнопок редактора в его нижней части см. в разделе "Редактирование ресурсов виджета".

Текстовые редакторы

Вы можете редактировать текстовый ресурс непосредственно в панели управления ресурсами, или можете щёлкнуть на ресурсе, чтобы отобразить текстовый редактор. Существует два текстовых редактора: один для однострочного текста и один для многострочного.

Всякий раз, щёлкнув на однострочном текстовом ресурсе на панели управления ресурсами (напр., ресурсе "Text String" для "PtText"), Вы увидите текстовый редактор:



Когда Вы выберете какой-либо многострочный текстовый ресурс – такой как ресурс "Text String" виджета PtLabel или PtMultiText – Вы увидите многострочный текстовый редактор:

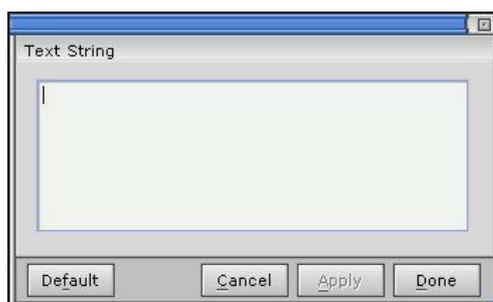


Рис. 6-7. Многострочный текстовый редактор

Одно- и многострочный редакторы похожи – вот их общие операции:

ДЛЯ ТОГО, ЧТОБЫ:	ДЕЛАЙТЕ ЭТО:
Удалить символ перед текстовым курсором	Нажмите <Backspace>
Удалить символ после курсора	Нажмите
Удалить несколько символов одновременно	Протащите указатель по символам, затем нажмите
Удалить целую строку	Нажмите <Ctrl>+<U>
"Перескочить" курсором на любую позицию в строке	Щёлкните на этой позиции
Переместить курсор с символа на символ	Нажмите <<-> или <->>
Переместить курсор в начало или конец строки	Нажмите <Home> или <End>

Для однострочного текстового редактора:

ДЛЯ ТОГО, ЧТОБЫ:	ДЕЛАЙТЕ ЭТО:
Обработать текстовую строку	Нажмите <Enter> или щёлкните на "Done" или "Apply"

Для многострочного текстового редактора:

ДЛЯ ТОГО, ЧТОБЫ:	ДЕЛАЙТЕ ЭТО:
Ввести новую строку текста	Нажмите <Enter>
Переместить курсор в начало или конец строки	Нажмите <Home> или <End>
Переместить курсор в начало или конец текста	Нажмите <Ctrl>+<Home> или <Ctrl>+<End>
Принять все изменения и закрыть редактор	Нажмите <Ctrl>+<Enter>

☞ Если Вам необходимо набрать символы, которых нет на клавиатуре, Вы можете использовать формирующие последовательности, описанные в разделе "Формирующие последовательности в Photon" приложения "Поддержка многоязычности Unicode".

Описание стандартных кнопок редактора в его нижней части см. в разделе "Редактирование ресурсов виджета".

Редактор функций

Когда Вы выбираете ресурс функции, такой как ресурс "Draw Function" (Pt_ARG_RAW_DRAW_F) виджета PtRaw, Вы увидите редактор функций:

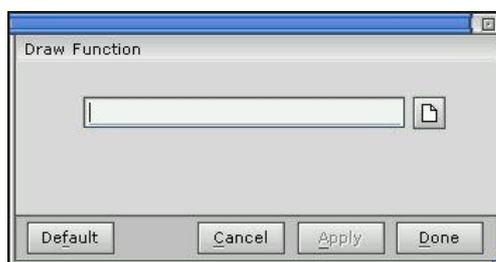


Рис. 6-8. Редактор функций

☞ Виджет должен иметь уникальное имя экземпляра, прежде чем Вы сможете редактировать его ресурсы функции. Наберите имя функции – см. раздел "Имена функций и файлов" в главе "Работа с программным кодом". Если Вы уже дали имя Вашему приложению при его сохранении (см. раздел "Сохранение приложения" в главе "Работа с приложениями"), Вы можете редактировать функцию, щёлкнув на кнопке справа от текстовой области.

Описание стандартных кнопок редактора в его нижней части см. в разделе "Редактирование ресурсов виджета".

Ответные реакции

Ответные реакции формируют связь между интерфейсом приложения и программным кодом Вашего приложения. Например, скажем, Вы хотите, чтобы приложение выполнило некое действие, когда пользователь выбрал определённую кнопку. В этом случае Вы прикрепите функцию ответной реакции к ответной реакции кнопки "Activate". Когда пользователь выбирает кнопку, виджет вызывает функцию ответной связи и Ваше приложение выполняет соответствующее действие программного кода ответной реакции.

Почти все виджеты поддерживают несколько типов ответных реакций. Эти ответные реакции могут быть заданы для виджета или быть унаследованы из его родительского класса. Некоторые из этих типов (определённые в виджете PtBasic), определены в следующей таблице:

ТИП	РЕСУРС	ОБЫЧНО ВЫЗЫВАЕТСЯ, КОГДА ПОЛЬЗОВАТЕЛЬ:
Activate	Pt_CB_ACTIVATE	Нажимает и отпускает левую кнопку мыши
Arm	Pt_CB_ARM	Нажимает левую кнопку мыши
Disarm	Pt_CB_DISARM	Отпускает левую кнопку мыши
Repeat	Pt_CB_REPEAT	Удерживает левую кнопку мыши нажатой
Menu	Pt_CB_MENU	Нажимает правую кнопку мыши

Для получения более полной информации по этим ответным реакциям см. "Справочник виджетов". Если Вы интересуетесь тем, как использовать Pt_CB_MENU для отображения модуля меню, см. главу "Доступ к модулям PhAB из программного кода".

Все виджеты Photon'a наследуют два других типа ответных реакций:

Ответные реакции "горячих клавиш"

Прикрепляют программный код ответной реакции к клавише или комбинации клавиш. Когда окно приложения получает фокус, "горячие клавиши" становятся активными. Нажатие любой такой клавиши вызывает исполнение программного кода, связанного с соответствующей клавишей.

Обработчики событий (Необработанные или отфильтрованные ответные реакции):

Прикрепляет ответные реакции непосредственно к событиям Photon'a.

В средах разработок некоторых оконных систем Вы можете присоединить к ответным реакциям виджета только программный код функции ответной реакции. Но используя для создания ответной реакции PhAB, Вы всегда можете сделать на один шаг больше, присоединяя окна, диалоги, меню и многое другое. Как упоминалось выше, эта расширенная функциональность обеспечивается в PhAB специальной формой ответной реакции, называемой привязанной ответной реакцией.

Привязанная ответная реакция также позволяет Вам добавить функциональности, недоступной, если Вы прикрепляете ответные реакции "вручную". Например, если Вы связали диалог с виджетом кнопки, Вы можете задать, где диалог возникнет. Вы сможете также задать установочную функцию, вызываемую автоматически перед выполнением диалога, после его выполнения, или и то и другое.

PhAB предоставляет две основные категории привязанных ответных реакций:

Привязанные ответные реакции модульного типа,

позволяющие Вам прикрепить модуль приложения к любой ответной реакции виджета. PhAB предлагает следующие категории ответных реакций модульного типа:

- Диалог
- Окно
- Меню
- Картинка

Для получения более полной информации см. раздел "Модульные ответные реакции" ниже в этой главе.

Привязанные ответные реакции кодового типа

позволяют Вам запускать функцию в программном коде при вызове ответной реакции виджета. PhAB предлагает следующие категории ответных реакций кодового типа:

- Код
- "Done"
- "Cancel"

Типы "Done" и "Cancel" обеспечивают дополнительную возможность: они будут автоматически закрывать или уничтожать родительский модуль виджета после вызова функции ответной реакции. Вы найдёте полезными эти типы для создания любых кнопок, закрывающих диалоговое окно.

☞ Ответная реакция "Done" в базовом окне завершает работу приложения; ответная реакция "Cancel" в базовом окне закрывает окно приложения, но не завершает работу приложения.

Для получения более полной информации см. раздел "Кодовые ответные реакции" ниже в этой главе.

Редактирование ответных реакций:

Редактор ответной реакции позволяет Вам добавлять, изменять, удалять или просматривать список ответных реакций виджета.

Чтобы узнать, как добавлять ответную реакцию в пункт команды или переключать пункт меню, см. раздел "Модули меню" в главе "Работа с модулями".

☞ Если Вы добавляете привязанную ответную реакцию к виджету, виджет должен иметь уникальное имя экземпляра. Если PhAB скажет Вам, что имя неуникально, используйте область "Widget Instance Name" на панелях управления ресурсами или ответными реакциями, чтобы отредактировать имя. Чтобы открыть редактор ответной реакции и отредактировать список ответных реакций виджета:

1. Выберите виджет, затем, если необходимо, переключитесь на панель управления ответными реакциями.
2. Выберите тип ответной реакции из списка ответных реакций виджета (например, чтобы добавить ответную реакцию Pt_CB_ACTIVATE, щёлкните на "Activate").

Вот простая сессия редактора ответных реакций:

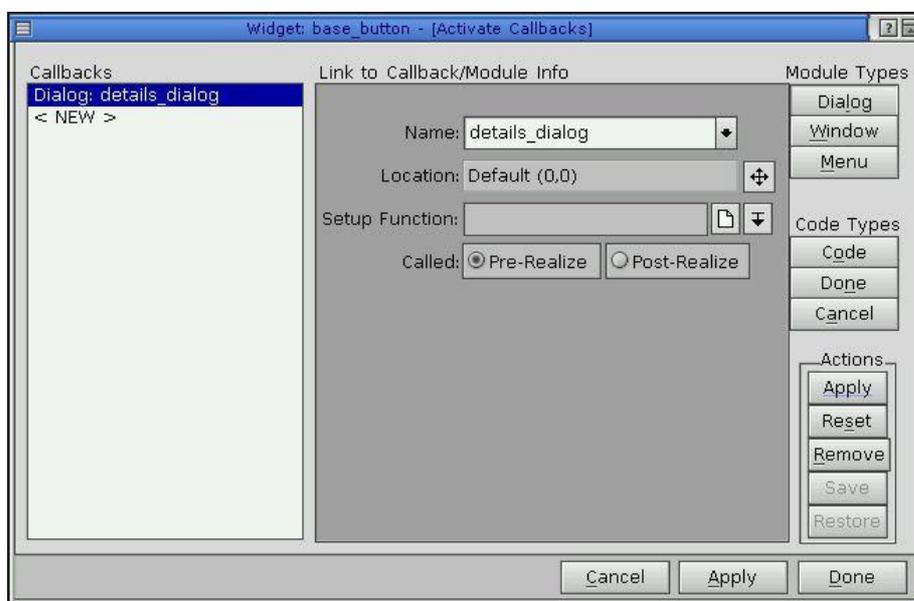


Рис. 6-9. Редактор ответных реакций

1. Чтобы добавить новую ответную реакцию, щёлкните на <NEW>. Для редактирования существующей ответной реакции щёлкните на этой ответной реакции в списке ответных реакций.
2. Если Вы добавляете новую ответную реакцию, выберите тип ответной реакции, которую Вы хотите добавить. Чтобы сделать это, выберите его из "Module Types" или "Code Types".
3. Заполните данные в секции "Link to Callback/Module Info". Области в этой секции зависят от типа выбранной ответной реакции. Для получения более полной информации см. разделы в этой главе, описывающие:
 - модульные ответные реакции
 - кодовые ответные реакции
 - ответные реакции "горячих клавиш"
 - обработчики событий (необработанные и отфильтрованные ответные реакции)
4. После того как Вы добавили или отредактировали какую-либо из ответных реакций, щёлкните на соответствующей кнопке:

- "Apply" – принять любые изменения; убедитесь, что сделали это перед тем, как начать работу с другими ответными реакциями.
- "Reset" – восстановить оригинальные значения для всей информации поответным реакциям.
- "Remove" – удалить ответную реакцию из списка ответных реакций.

Модульные ответные реакции

Для связи виджета с модулем могут использоваться привязанные ответные реакции модульного типа. Например, выбор кнопки будет приводить к созданию модуля.

☞ Когда Вы используете для создания модуля привязанную ответную реакцию модульного типа, модуль становится потомком базового окна Вашего приложения, а не потомком модуля, содержащего виджет, для которого определена привязанная ответная реакция.

Если Вы хотите, чтобы родителем модуля было что-либо иное, чем базовое окно, Вам необходимо использовать внутреннюю связь в программном коде Вашего приложения, чтобы создать модуль. Для получения более полной информации по внутренним связям и другим случаям, когда Вы это используете, см. главу "Доступ к модулям PhAB из программного кода".

В зависимости от вида ответной реакции модульного типа, которую Вы создаёте, редактор ответных реакций PhAB отображает все или некоторые из этих полей:

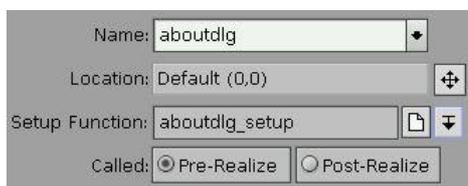


Рис. 6-10. Области редактора ответных реакций

Name	Имя модуля. Если Вы щёлкните на иконке рядом с этой областью, то увидите список существующих модулей. Либо выберите имя из этого списка, либо введите имя модуля, которое не существует (PhAB создаёт модуль для Вас, когда Вы добавляете ответную реакцию).
Location	Позволяет задавать, где модуль должен отображаться. По умолчанию модуль меню располагается под виджетом, который этот модуль вызвал. Для всех остальных модулей принимаемое по умолчанию расположение определяется оконным менеджером. Для получения более полной информации см. раздел "Позиционирование модуля" в главе "Работа с модулями".
Setup Function	Позволяет Вам задать функцию, которая может быть вызвана в два различных момента времени (как задано в области "Called"): <ul style="list-style-type: none">• перед тем как модуль отображается (предреализация)• после того как модуль отобразился (постреализация). Вы можете задать только одну установочную функцию – PhAB API вызывает одну и ту же функцию как при пред-, так и при постреализации модуля. Для того чтобы отличить, какой проход функции был вызван, проверьте вызывавший её программный код. Щёлкните на иконке возле области "Setup Function", чтобы отредактировать функцию или выбрать её из существующих ответных реакций.
Hotkey	(только для ответных реакций "горячих клавиш"). Клавиатурная клавиша и модификатор (такой как <Alt> или <Ctrl>), которые запускают ответную реакцию. См. раздел "Задание ответных реакций "горячих клавиш".

Event Mask (только для обработчиков событий). Позволяет Вам задавать, к каким событиям Photon'a виджет будет чувствителен. См. "Обработчики событий – необработанные и отфильтрованные ответные реакции".

Предреализационная установочная функция

Предреализационная установочная функция позволяет Вам предустановить модуль. Например, пусть для Вашего приложения необходимо "заполнение чистых полей" диалога перед отображением этого диалога. В функции установки Вы можете использовать генерируемые PhAB'ом имена деклараций для предустановки ресурсов различных виджетов диалога.

После того как установочная функция отработает, она возвращает Pt_CONTINUE. Диалог затем реализуется и отображается на экране, используя все предустановленные значения.

Постреализационная установочная функция

Постреализационная установочная функция работает во многом как предреализационная функция установки, за исключением того, что она вызывается после того, как диалог отобразится на экране. Обычно этот тип функции используется, когда Вам необходимо обновить модуль после того, как он стал видимым. Диалог генерации кода PhAB'a является хорошим примером такого случая. Он отображает экран и затем, используя постреализационную функцию, обновляет шкалу прогресса на протяжении того, как генерируется программный код приложения.

☞ Функция установки для модуля меню вызывается только перед тем, как меню отображается. Для большинства приложений, Вам понадобится использовать эту функцию для того, чтобы установить начальное состояние пунктов меню. Например, Вы можете использовать эту функцию, чтобы отключить определённые пункты меню перед тем, как меню отобразится.

Установочные функции хранятся в заготовочных файлах

Когда Вы задаёте установочную функцию, PhAB генерирует заготовочную функцию; для получения более полной информации по заданию языка (C или C++) и имени файла, см. раздел "Имена функций и имена файлов" в главе "Работа с программным кодом".

Ответные реакции кодового типа

Этот тип ответной реакции позволяет Вам при вызове ответной реакции виджета запускать на выполнение функцию кодового типа.

☞ Вы можете добавить кодовые ответные реакции из программного кода своего приложения, но проще делать это в PhAB. Для получения более полной информации см. раздел "Ответные реакции" в главе "Управление виджетами в программном коде приложения".

Когда Вы создаёте привязанную реакцию кодового типа, редактор ответных реакций предложит Вам определить следующее:

Function Это функция, которая будет вызываться, когда виджет вызывает ответную реакцию. Для типов "Done" или "Cancel" эта функция необязательна, так что Вы можете присоединить ответную реакцию, просто закрыв модуль. Как сказано выше, "Done" и "Cancel" являются похожими, за исключением того, что ответная реакция "Done" в базовом окне завершает работу приложения, тогда как ответная реакция "Cancel" закрывает окно, но не завершает приложение. В действительности нет разницы между функциями ответных реакций "Done" и "Cancel" – они просто задействуют различный программный код в ответной реакции. Например, скажем, что Вы имеете диалог с кнопками "Done" и "Cancel".

Если Вы присоедините ответную реакцию типа "Done" к кнопке "Done" и ответную реакцию типа "Cancel" к кнопке "Cancel", Вы сможете использовать одну и ту же функцию в программном коде в обоих случаях и просто смотреть на исполняемый код, чтобы определить, какая кнопка выбрана пользователем.

Горячая клавиша (только для ответных реакций горячих клавиш)

Клавиатурная клавиша и модификатор (такой как <Alt> или <Ctrl>), которые запускают ответную реакцию. См. раздел "Ответные реакции горячих клавиш".

Маска событий (только для обработчиков событий)

Позволяет Вам задавать, какие события Photon'a являются чувствительными для виджета. См. "Обработчики событий – необработанные и отфильтрованные ответные реакции".

Функции ответных реакций хранятся в заготовочных файлах

Когда Вы задаёте функцию ответной реакции, PhAB генерирует заготовку функции; для получения информации по заданию языка (C или C++) и имени файла, см. раздел "Имена функций и файлов" в главе "Работа с программным кодом".

Ответные реакции горячих клавиш

Виджеты поддерживают ответные реакции горячих клавиш. Эти ответные реакции позволяют Вам присоединить клавиши клавиатуры к заданным функциям ответных реакций. Когда окно приложения получает фокус, горячие клавиши становятся активными. Нажатие такой клавиши вызывает соответствующую связанную с горячей клавишей ответную реакцию.

Этот раздел включает:

- Горячие клавиши – основы
- Задание метки горячей клавиши
- Задание ответной реакции
- Обработка горячей клавиши
- Отключение ответной реакции

Горячие клавиши – основы

Вот некоторая основная информация о горячих клавишах:

- Горячие клавиши – это комбинация клавиши символа и ключа-модификатора (<Alt>, <Shift> или <Ctrl>). По большей части для горячих клавиш используется <Alt>.
- Вы можете использовать в качестве горячей клавиши модификатор сам по себе, однако это, по-видимому, не является хорошей идеей.
- Горячая клавиша не вызывается, если заблокирован какой-либо прародитель виджета, которому она принадлежит.
- Горячая клавиша обрабатывается после того, как виджет получает событие клавиши. Если виджет поглощает событие, никакая ответная реакция горячей клавиши не вызывается. Так, когда текстовая область получает фокус, клавиша <Enter>, клавиши стрелок, клавиша пробела и все отображаемые символы не работают как горячие клавиши, поскольку виджет поглощает эти события. Это обычно является требуемым поведением (предполагать редактирование в приложении, все клавиши стрелок в котором определены как горячие).

Вы можете заставить обрабатывать горячие клавиши прежде получения события, установив `Pt_HOTKEYS_FIRST` в ресурсе `Pt_ARG_CONTAINER_FLAGS` контейнера виджета (окна, панели, ...), содержащего виджеты, которые обычно поглощают возможные события горячих клавиш. Установка этого флага в окне гарантирует, что все горячие клавиши будут обработаны до того, как какой-либо виджет получит событие нажатия клавиши. Для получения более полной информации см. ниже раздел "Обработка горячих клавиш".

- Чтобы горячие клавиши виджетов были активны, эти виджеты должны быть выбираемы (за исключением расчленённых виджетов, таких как окна или меню). Убедитесь, что в ресурсах `Pt_ARG_FLAGS` виджетов установлены флаги `Pt_SELECTABLE` и `Pt_GETS_FOCUS`.

Если виджеты обычно не являются выбираемыми и Вы не желаете, чтобы их внешний вид изменился при выборе, Вы должны также установить флаг `Pt_SELECT_NOREDRAW` виджета.

- Часто не имеет значения, с каким виджетом связана ответная реакция. В этих случаях просто присоедините ответную реакцию к окну.

Задание метки горячей клавиши

Установки горячей клавиши недостаточно – Вам необходимо сообщить пользователю об этом. Вы должны отобразить метку горячей клавиши на виджете, вызываемом горячей клавишей:

- Для большинства виджетов отредактируйте ресурс клавиши быстрого доступа (`Pt_ARG_ACCEL_KEY`). Задайте символ в метке виджета, который Вы хотите подчеркнуть. Вы не можете включить в метку какую-либо клавишу-модификатор.
- Для пунктов меню подчеркнутый символ является кнопкой быстрого доступа, которую Вы можете использовать для выбора пункта, когда отображено меню. Метка горячей клавиши отображается отдельно, справа от метки пункта меню. Задайте горячую клавишу (включая клавиши-модификаторы) в области "Accel Text" редактора меню.

Задание ответной реакции

В PhAB каждый список ответных реакций виджета отображает вход, называемый "Hotkey" или `Pt_CB_HOTKEY`, который используется для определения горячих клавиш. Перед тем как определить горячую клавишу, Вам необходимо определить, где это сделать. Где Вы определите ответную реакцию горячей клавиши, зависит от того:

- где должен появиться модуль (такой как меню)
- какой виджет Вам нужен в функции ответной реакции
- куда пойдёт пользователь, чтобы нажать горячую клавишу

Где должен появиться модуль

Когда Вы определяете горячую клавишу, Вы можете задать, где появиться модулю. Например, если горячая клавиша предназначена для отображения модуля меню, связанного с виджетом `PtModuleButton` в Вашем `PtMenuBar` окна, определите горячую клавишу в кнопке меню. Используйте "Location dialog", чтобы меню появилось под кнопкой меню. Для получения более полной информации см. раздел "Позиционирование модуля" в главе "Работа с модулями".

Какой виджет Вам нужен в функции ответной реакции

Виджет, имеющий ответную реакцию, является подходящим для функции ответной реакции.

Куда идти пользователю, чтобы нажать горячую клавишу

Например, если горячая клавиша является быстрым входом в пункт меню, добавьте горячую клавишу к окну, в котором меню используется, а не к модулю меню.

- ☞ Горячие клавиши в данном модуле должны быть уникальны. Если Вы определили некую клавишу более одного раза, используется последнее определение. Если Вы разрабатываете многоязычное приложение, Вам понадобятся различные наборы горячих клавиш для каждого языка. См. главу "Поддержка международных языков".

Когда Вы выбираете ответную реакцию `Pt_CB_HOTKEY`, всплывает редактор ответных связей с областью "Hotkey" в зоне связанной информации:



Рис. 6-11. Область "Hotkey" в редакторе ответных реакций

При создании ответных реакций горячей клавиши Вы должны заполнить область "Hotkey". Есть два пути установить горячую клавишу: один простой, другой не очень.

- «не такой уж простой способ» – Вы можете набрать значение горячей клавиши в шестнадцатеричном формате в области "Hotkey". Чтобы найти значение для клавиши, которую Вы хотите использовать, посмотрите хедер-файл `<photon/PkKeyDef.h>` и найдите имя клавиши, предварённое префиксом `Pk_`.
- ☞ Используйте для горячих клавиш прописные буквы; заглавные не будут работать. Например, для горячей клавиши `<Alt>+<F>` смотри шестнадцатеричное значение не для `Pk_F`, а для `Pk_f`.

Область имеет также три переключающиеся кнопки – "Ctrl", "Shift" и "Alt", позволяющие Вам задать модификатор для горячей клавиши.

- «простой способ» – нажмите кнопку справа от кнопки-переключателя "Alt", затем нажмите сочетание клавиш, которое Вы хотите использовать в качестве горячей клавиши. PhAB автоматически определит клавишу и модификатор, которые Вы нажали.

Обработка горячих клавиш

Вот как работает горячая клавиша:

- когда событие клавиши достигает окна, окно направляет событие своим порождённым виджетам;
- если порождение это событие поглощает, ничего больше не происходит;
- в противном случае событие проверяется в списке горячих клавиш окна. Если горячая клавиша найдена, вызывается ответная реакция;
- если ответная реакция не найдена, просматривается список горячих клавиш родительского окна, и далее вверх по иерархии окон.

Ресурс `Pt_ARG_CONTAINER_FLAGS` виджетов контейнерного класса включает несколько флагов, оказывающих влияние на обработку горячих клавиш:

`Pt_HOTKEY_TERMINATOR`

Не допускает проход поиска горячей клавиши вверх к родительскому контейнеру. Флаг `Pt_HOTKEY_TERMINATOR` работает, только если он установлен в разобранном виджете контейнерного класса.

`Pt_HOTKEYS_FIRST`

Обработка событий клавиш, достигших контейнера, как горячих клавиш, перед тем как они проходят к потомкам контейнера. Если событие является горячей клавишей, оно поглощается, т.е. не проходит к потомкам.

Отключение горячих клавиш

Выдача пользователю визуальной индикации, что горячая клавиша отключена, отличается от действительного отключения горячей клавиши.

Для того, чтобы выдать визуальную индикацию, используйте технологию, соответствующую виджету:

- если горячая клавиша присоединена к кнопке, установите флаг Pt_GHOST и снимите флаги Pt_SELECTABLE и Pt_GETS_FOCUS в ресурсе кнопки Pt_ARG_FLAGS;
- если горячая клавиша присоединена к пункту меню, созданному в PhAB, вызовите `ApModifyItemState()`;
- ...

Чтобы отключить горячую клавишу, используйте одну из следующих технологий:

- Не отключайте горячую клавишу. Вместо этого, в качестве первого, что должно быть выполнено в программном коде ответной реакции горячей клавиши, введите проверку на нечто, что должно быть сделано. Если это не сделано, просто вернитесь из ответной реакции. Например, если ответные реакции горячей клавиши – это "вклеить" текст, проверьте, есть ли что "вклеивать". Если нечего, просто вернитесь
или
- За исключением разобранных виджетов, если виджет, к которому присоединена ответная реакция горячей клавиши, не является выбираемым, горячая клавиша обрабатывается так, как будто она не существует. Чтобы виджет был выбираемым, должен быть установлен флаг Pt_SELECTABLE в ресурсе Pt_ARG_FLAGS.

Хорошим основанием для такого подхода является то, что это работает, даже если Ваше приложение имеет одну и ту же горячую клавишу, описанную более чем в одном окне. Например, мы можем иметь меню "Edit" в базовом окне и кнопку "Erase" в порождённом окне, оба с сочетанием <Alt>+<E> в качестве горячей клавиши. Если в текущий момент фокус имеет порождённое окно и пользователь нажимает <Alt>+<E>, вызывается ответная реакция кнопки "Erase" порождённого окна.

Теперь, если мы отменим кнопку "Erase" в порождённом окне, мы хотим использовать <Alt>+<E> для того, чтобы появилось меню "Edit" базового окна. В этом сценарии, пока кнопка "Erase" является выбираемой, будет вызываться её ответная реакция. Поэтому мы просто делаем кнопку "Erase" невыбираемой. Теперь, когда пользователь нажимает <Alt>+<E>, появляется меню "Edit" базового окна, даже при том, что порождённое окно ещё имеет фокус.

или

- Вы можете вызвать `PtRemoveHotkeyHandler()`, чтобы удалить горячую клавишу, и впоследствии вызвать `PtAddHotkeyHandler()`, чтобы включить её снова.

Обработчики событий – необработанные и отфильтрованные ответные реакции

Обработчики событий позволяют Вам реагировать непосредственно на события Photon'a. Вы можете присоединить обработчики событий к любому виджету; они похожи на другие ответные реакции виджета, но с дополнением в виде маски событий. Используя эту маску, Вы можете выбрать, какие события будут получать Ваши ответные реакции.

Вы найдёте это крайне полезным для получения событий Pt_EV_DRAG для конкретного окна. Для получения более подробной информации по перетаскиванию см. раздел "Перетаскивание" в главе "События".

Pt_Widget определяет следующие ресурсы обработки событий:

- Pt_CB_FILTER Вызывается перед тем, как событие достигло виджета
- Pt_CB_RAW Вызывается после того, как виджет обработал событие (даже если виджет поглотил событие)

➤ Описание необработанных и отфильтрованных обработчиков событий и их использования см. в разделе "Обработчики событий – необработанные и отфильтрованные ответные реакции" в главе "События".

Для получения более подробной информации о добавлении обработчиков событий в программный код приложения см. "Обработчики событий" в главе "Управление виджетами в программном коде приложения".

Чтобы присоединить необработанную или отфильтрованную ответную реакцию:

1. Выберите виджет, затем переключитесь, если необходимо, в панель управления ответными реакциями.
2. Щёлкните на ресурсе Pt_CB_RAW (необработанные события) или Pt_CB_FILTER (отфильтрованные), чтобы открыть редактор ответных реакций.
3. Всплывёт редактор с областью "Event Mask" в зоне связанной информации:

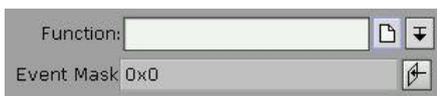


Рис. 6-12. Область маски событий в редакторе ответных реакций

Область маски событий позволяет Вам задавать, к каким событиям Photon'a должен быть чувствителен виджет. Если случается любое из этих низкоуровневых событий, виджет вызывает ответную реакцию.

Щёлкните на иконке рядом с этой областью, чтобы открыть селектор событий:

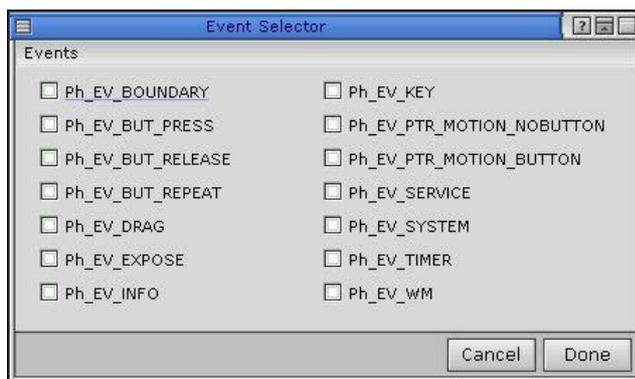


Рис. 6-13. Селектор событий

4. Выберите события, к которым должен быть чувствителен виджет, затем закройте селектор.

Для получения более полной информации см. типы событий, описанные в структуре PhEvent_t в "Справочнике библиотеки Photon'a".

Глава 7. Управление геометрией

Эта глава описывает, как задать или точно регулировать геометрию Ваших виджетов. Она включает:

- Контейнер виджетов
- Согласование геометрии
- Абсолютное позиционирование
- Выравнивание виджетов с использованием групп
- Удерживающее управление с использованием анкером
- Удерживание позиции или ограничение изменения размеров без анкером

Контейнер виджетов

Контейнерный виджет является порождением виджетного класса `PtContainer`. Контейнерными виджетами являются только те виджеты, которые могут иметь потомков. Любой виджет, не имеющий окна в своём владении, всегда визуализируется внутри границ своего родителя. Только виджеты, принадлежащие к производному классу от виджетного класса `PtWindow`, получают в свою собственность окно.

Контейнерные виджеты отвечают за выполнение управления геометрией. Первой обязанностью контейнерного виджета является позиционирование каждого потомка и установка его размеров так, чтобы обеспечить желаемое расположение для всех своих потомков. Контейнер также может накладывать на своих потомков ограничения по размерам (например, заставляет их все быть одного и того же размера). Кроме того, контейнер должен ограничивать потомков так, чтобы они не появлялись вне границ контейнера. Обычно это достигается путём обрезания потомков.

Для понимания того, как различные контейнеры выполняют управление геометрией, важно понимать геометрию виджета. См. "Геометрия виджета" во введении к настоящему руководству.

Согласование геометрии

При реализации виджета во всех виджетах иерархии семейства виджета запускается процесс согласования геометрии. Каждому потомку виджета даётся благоприятная возможность просчитать свои размеры. Это проходит волной вниз через все виджеты семейства, в результате чего пересчитываются размеры каждого потомка.

Как только каждый потомок пересчитал свои желаемые размеры, родительский виджет может попытаться определить расположение для своих потомков. Компоновка, выполняемая виджетом, зависит от:

- политики компоновки виджетов;
- любого размера, установленного для виджета;
- размерами и желаемой позицией для каждого из потомков.

Если приложение задаёт размер для виджета, то он может выбирать, как располагать потомков, используя только это доступное пространство. На это влияет политика изменения размеров, установленная для виджета. Ресурс `Pt_ARG_RESIZE_FLAGS` устанавливает флаги, определяющие политику изменения размеров для виджета. Флаги задают в отдельности политику изменения ширины и высоты виджета. Если для какого-либо из измерений политика не определена, виджет не пытается изменять свои размеры а этом измерении при выполнении компоновки. Любая другая

политика изменения размеров позволяет виджету увеличиваться в этом измерении, чтобы приспособиться под своих потомков. Более детально это описано в разделе "Политика изменения размеров" ниже.

Если виджет не имеет предопределённых размеров, он пытается изменить свои размеры, чтобы приспособиться по всем потомкам, используя соответствующую политику компоновки. Таким образом, он вначале пытается определить правильную компоновку и затем определяет пространство, необходимое для подгонки под эту компоновку.

Процесс компоновки определяет желаемое местоположение каждого потомка. Политика компоновки, используемая для виджета, управляет, как процесс компоновки пытается позиционировать потомков один относительно другого. Она должна принимать в расчёт размеры потомков. Контейнер отвечает за фиксирование позиции каждого потомка, так что политика компоновки может выбирать, принимать или нет во внимание атрибуты позиции потомков.

Выполняя компоновку, виджет может также принимать во внимание политику изменения размеров. Основываясь на этой политике, он определяет, должен ли он корректировать их ширину или высоту, или же изменять компоновку, что объясняется ограничением пространства. Виджет пытается выбрать компоновку, которая бы наилучшим образом совмещала ограничения, налагаемые какими-либо ограничениями размеров и политикой компоновки. После определения желаемой позиции для каждого из своих потомков виджет вычисляет ширину и высоту, требуемые им для размещения потомков в этих местоположениях. Он изменяет при необходимости их размеры, чтобы приладить каждый из потомков в желаемой позиции. Если это невозможно, потому что этого не позволяет политика изменения размеров, виджет пересчитывает позицию, чтобы пристроить потомков внутри доступного свободного места.

После того как компоновка успешно установлена, виджет устанавливает позицию для каждого потомка путём внесения изменения в атрибуты позиции потомков.

Политика изменения размеров

Любые изменения виджета, которые могут оказать действие на объём пространства, требуемого для отображения его содержания, могут приводить к изменению размеров самого этого виджета, с тем чтобы вместить его содержание. На это влияет политика изменения размеров, назначенная виджету.

Политика изменения размеров действует на базовые виджеты и на контейнеры. Контейнер проверяет свою политику изменения размеров при компоновке своих потомков, чтобы определить, будет ли он изменять свои размеры, чтобы разместить всех потомков в их желаемом местоположении. Во время процесса согласования геометрии это действие распространяется вверх по семейству виджетов до тех пор, пока не определяется размер виджета окна.

Политикой изменения размеров управляет ресурс `Pt_ARG_RESIZE_FLAGS`. Этот ресурс состоит из отдельного набора флагов для ширины и высоты. Значение флагов определяет условия, в которых виджет пересчитывает соответствующий размер. Значения проверяются каждый раз, когда виджет реализуется или изменяется его содержание.

☞ Если политика изменения размеров конфликтует с анкерами, `Pt_ARG_RESIZE_FLAGS` переписывает `Pt_ARG_ANCHOR_OFFSETS` и `Pt_ARG_ANCHOR_FLAGS`.

Имеются следующие флаги изменения размеров:

`Pt_RESIZE_X_ALWAYS`

Пересчитывает размер виджета всякий раз, когда изменяется значение размера *x*. Виджет растёт или сжимается в горизонтальном направлении в соответствии с изменением его содержания.

Например, следующий рисунок показывает кнопку с установленным флагом `Pt_RESIZE_X_ALWAYS` при изменении надписи с "Hello" к "Hello, world" и к "Hi".



`Pt_RESIZE_Y_ALWAYS`

Пересчитывает размер виджета всякий раз, когда изменяется значение размера `y`. Виджет растёт или сжимается в вертикальном направлении в соответствии с изменением его содержания.

`Pt_RESIZE_XY_ALWAYS`

Пересчитывает размер виджета всякий раз, когда изменяется значение размера `x` или `y`. Виджет растёт или сжимается в обоих направлениях в соответствии с изменением его содержания.

☞ Флаг `Pt_RESIZE_XY_ALWAYS` в PhAB не определён. Он предоставляется для Вашего удобства, когда установка флагов изменения размеров осуществляется из Вашего программного кода.

`Pt_RESIZE_X_AS_REQUIRED`

Пересчитывает размер виджета всякий раз при изменении размера `x` и не производит подгонку при наличии свободного пространства. Например, следующий рисунок показывает кнопку с установленным флагом `Pt_RESIZE_X_AS_REQUIRED` при изменении надписи с "Hello" к "Hello, world" и к "Hi".



`Pt_RESIZE_Y_AS_REQUIRED`

Пересчитывает размер виджета всякий раз при изменении размера `y` и не производит подгонку при наличии свободного пространства.

`Pt_RESIZE_XY_AS_REQUIRED`

Пересчитывает размер виджета всякий раз при изменении размера `x` или `y` и не производит подгонку при наличии свободного пространства.

☞ Флаг `Pt_RESIZE_XY_AS_REQUIRED` в PhAB не определён. Он предоставляется для Вашего удобства, когда установка флагов изменения размеров осуществляется из Вашего программного кода.

Эти флаги также могут быть модифицированы значениями другого набора флагов, а именно

- `Pt_RESIZE_X_INITIAL`
- `Pt_RESIZE_Y_INITIAL`
- `Pt_RESIZE_XY_INITIAL`

☞ Флаг `Pt_RESIZE_XY_INITIAL` в PhAB не определён. Он предоставляется для Вашего удобства, когда установка флагов изменения размеров осуществляется из Вашего программного кода.

Если Вы установили какой-либо из этих "инициализирующих" флагов, виджет не будет изменять свои размеры в ответ на изменение данных – он изменяет свои размеры только в процессе согласования геометрии всякий раз, когда реализуется. Виджет либо делает себя в точности по размерам своего содержимого, либо увеличивается в размерах, чтобы разместить своё содержимое, если его размеры в этот момент недостаточно велики.

Если никакой из флагов изменения размеров не установлен, виджет не пытается рассчитать свои собственные размеры, а использует те размеры, которые были установлены приложением (таким образом, возможно, обрезая в результате содержимое виджета).

Например, следующий рисунок показывает кнопку, у которой не установлены никакие флаги изменения размеров, при изменении надписи с "Hello" к "Hello, world" и к "Hi":



Установка политики изменения размеров в PhAB

Вы можете установить эти флаги в PhAB при редактировании флагов изменения размеров контейнера Pt_ARG_RESIZE_FLAGS, как показано ниже:



Установка политики изменения размеров в программном коде приложения

Вы можете также установить флаги изменения размеров контейнера в Вашем программном коде, используя метод, описанный в главе "Управление ресурсами в программном коде приложения".

Предоставляются битовые маски, позволяющие управлять путём установки битов. Имеется по одной битовой маске для политики изменения размеров по x и по y:

- Pt_RESIZE_X_BITS
- Pt_RESIZE_Y_BITS
- Pt_RESIZE_XY_BITS

Например, чтобы сделать контейнер вырастающим для размещения всех своих потомков, если его размеры при реализации виджета недостаточно велики, установите оба флага установки размеров initial (начальный) и required (требуемый) по координатам x и y:

```
Pt_SetResource(ABW_my_container, Pt_ARG_RESIZE_FLAGS,
               (Pt_RESIZE_XY_INITIAL | Pt_RESIZE_XY_AS_REQUIRED),
               Pt_RESIZE_X_BITS | Pt_RESIZE_Y_BITS);
```

Чтобы установить список аргументов для очистки политики изменения размеров по x:

```
Pt_SetResource(ABW_my_container, Pt_ARG_RESIZE_FLAGS,
               Pt_FALSE, Pt_RESIZE_X_BITS);
```

Имеется также несколько констант, упрощающих установку этих флагов. Например, есть константа, представляющая битовую маску для установки одновременно флагов по x и по y, и есть константы для наложения изменений для каждого флага по координатам x или y. Все эти константы определены в хедер-файле <photon/PtWidget.h>

Абсолютное позиционирование

Самой основной формой компоновки, которую может обеспечить контейнер, является позиционирование своих потомков без наложения каких-либо ограничений на их размеры или позиционирование. В такой ситуации порождённый виджет закрепляется в своём собственном положении внутри контейнера, и контейнер не изменяет его размеров.

Виджет, использующий эту политику компоновки, является чем-то аналогичным доске объявлений. Вы можете прикрепить сообщение на доске объявлений и они остаются там, куда их прикрепили. Все контейнерные виджеты могут выполнять абсолютное позиционирование.

Простейшим путём позиционирования и установления размеров каждого потомка является использование мыши в PhAB.

Чтобы задать позицию каждого из потомков из программного кода Вашего приложения, Вы должны установить для каждого потомка ресурс `Pt_ARG_POS`. Если виджеты должны быть согласованы или быть предопределённых размеров, Вы также должны установить для каждого потомка ресурс `Pt_ARG_DIM`. Задаваемая Вами позиция является относительной верхнего левого угла холста родителя, так что при позиционировании потомков Вы можете не принимать во внимание границы родителя. По умолчанию все виджеты, позиционируемые абсолютно, используют политику изменения размеров `Pt_AS_REQUIRED` и `Pt_INITIAL`. Другими словами, начальные размеры контейнера выбираются при его реализации. Контейнер делается достаточно большим, чтобы разместить всех потомков в заданном местоположении и давая им возможность после реализации принять свои размеры.

Простейшим путём выполнения абсолютного позиционирования является размещение и позиционирование виджетов внутри главного виджета `PtWindow` приложения. Если Вам необходимо создать контейнерный виджет, выполняющий абсолютное позиционирование, как часть другого контейнера, Вы можете использовать виджет `PtContainer`.

Выравнивание виджетов с использованием групп

Виджеты `PtGroup` являются виджетами контейнерного класса, которые могут управлять геометрией своих потомков. Вы найдёте это полезным для выравнивания виджетов по горизонтали, вертикали или как матрицу. Они также обладают уникальной способностью растягивать порождённые виджеты.

PhAB расширяет полезность этого класса виджетов путём превращения их в ориентированную на действие команду "Group". Используя эту команду, Вы можете выбрать несколько виджетов внутри модуля и группировать их вместе в один групповой виджет. Если Вы попытаетесь выбрать любой виджет из группы, щёлкнув на нём, будет выбрана вся группа.

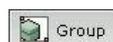
Когда Вы выбираете группу, панель управления ресурсами показывает ресурсы, имеющиеся в распоряжении виджетного класса `PtGroup`. Это включает ресурсы, позволяющие Вам выравнивать виджеты внутри группы и устанавливая эксклюзивное по выбору поведение.

Виджет `PtGroup` может быть использован для расстановки группы виджетов по строкам, колонкам или в виде таблицы. Для управления этим используется несколько ресурсов, и они интерпретируются несколько по-разному в зависимости от желаемой компоновки потомков.

Объединение виджетов в группу

Чтобы объединить виджеты в группу:

1. Выберите виджеты, используя либо метод ограничивающего прямоугольника, либо метод "Shift и щелчок"(как описано в главе "Создание виджетов в PhAB"). Вы можете использовать "Shift и щелчок", если собираетесь выравнивать виджеты по порядку, используя ресурс "Orientation". Первый выбранный Вами виджет становится первым внутри группы. Если порядок не важен или выравнивание не требуется, приятнее работать с методом ограничивающего прямоугольника.
2. Выполните одно из следующих действий:
 - Выберите пункт "Group Together" из меню "Edit".
 - Нажмите <Ctrl>+<G>
 - Щёлкните на кнопке "Group" на панели инструментов PhAB:



PhAB сгруппирует виджеты и выберет группу.

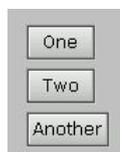
Получение доступа к виджетам в группе

Несмотря на то, что PhAB интерпретирует группу как один виджет, Вы по-прежнему можете получить доступ к любому отдельному виджету, входящему в группу. Чтобы это сделать, используйте клавиши "Следующая" и "Предыдущая" в панели управления ресурсами или ответными реакциями, или выберите виджет непосредственно из панели дерева модулей. Более подробно это описано в разделе "Выбор виджетов" в главе "Создание виджетов в PhAB".

Горизонтальное или вертикальное выравнивание виджетов

Ресурс ориентации Pt_ARG_GROUP_ORIENTATION управляет тем, выравниваются ли потомки группового виджета построчно или по колонкам. Значение Pt_GROUP_VERTICAL приводит к тому, что потомки выстраиваются вертикально, тогда как значение Pt_GROUP_HORIZONTAL – к тому, что они выстраиваются горизонтально. Вы можете управлять величиной свободного пространства, остающегося между виджетами, выстроенными в групповом виджете, путём использования ресурса Pt_ARG_GROUP_SPACING. Значение ресурса даёт число пикселей, оставляемых между виджетами.

Следующий пример показывает, как располагаются несколько потомков, если группа использует вертикальную ориентацию с пятью пикселями свободного пространства между потомками:



Если ориентация изменена на горизонтальную, группа выглядит так:

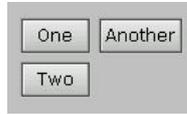


При первой реализации групповой виджет первоначально устанавливает свои размеры, чтобы после того, как все потомки будут расположены, быть достаточно большими, чтобы все их вместить.

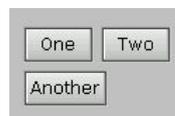
Выстраивание виджетов построчно и в колонки

Установкой значения ресурса `Pt_ARG_GROUP_ROWS_COLS` в величину, большую чем единица, можно использовать групповой виджет для размещения его потомков построчно и в колонки, создавая таблицы.

Интерпретация этого ресурса зависит от ориентации:



- Когда ориентация вертикальная, этот ресурс определяет число отображаемых строк; число колонок вычисляется на основе числа виджетов, чтобы получить верное число строк:
- В противном случае значение определяет число колонок, и виджет вычисляет число строк:



Последняя строка или колонка может иметь меньшее количество виджетов, чем остальные.

Когда элементы группы скомпонованы в строки и колонки, сами виджеты могут быть тесно упакованы или же они могут быть расставлены с равными интервалами построчно и/или колонками. Это управляется ресурсом `Pt_ARG_GROUP_SPACING`.

Использование флагов групп

Виджет `Pt_Group` включает набор флагов `Pt_ARG_GROUP_FLAGS`, которые могут быть использованы для управления тем, как порождённые виджеты могут быть выбраны, изменены в размерах и растянуты:

`Pt_GROUP_EXCLUSIVE`

Позволяет быть одновременно установленным только одному потомку. Этот флаг может использоваться, чтобы сделать группу кнопок-переключателей радиокнопками (т.е. установить взаимоисключающий выбор).

`Pt_GROUP_EQUAL_SIZE`

Размещает все виджеты в сетке, используя выбранный для группы размер ячейки, основанный на ширине самого широкого потомка и высоте самого высокого. Размеры всех потомков при компоновке устанавливаются в эти размеры.

`Pt_GROUP_EQUAL_SIZE_HORIZONTAL`

Делает все виджеты шириной, равной ширине самого широкого виджета.

`Pt_GROUP_EQUAL_SIZE_VERTICAL`

Делает все виджеты высотой, равной высоте самого высокого виджета.

`Pt_GROUP_NO_SELECT_ALLOWED`

Устанавливает этот флаг для исключяющей группы, если обоснованно не иметь никакого набора потомков. Пользователь может снять выбор текущего набора потомков, щёлкнув на нём ещё раз.

`Pt_GROUP_NO_KEYS`

Не позволяет пользователю перемещаться внутри группы с помощью клавиш со стрелками.

`Pt_GROUP_NO_KEY_WRAP_HORIZONTAL`

Не позволяет автоматически переходить на другую сторону группы при использовании клавиш с левой и правой стрелками (т.е. например, перейти в начало, достигнув конца. Прим. пер.).

`Pt_GROUP_NO_KEY_WRAP_VERTICAL`

Не позволяет автоматически переходить в вершину или вниз группы при использовании клавиш со стрелками вверх и вниз.

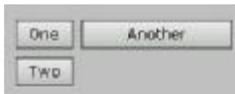
Pt_GROUP_STRETCH_VERTICAL

Растягивает нижнюю строку виджетов при расширении группы



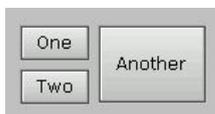
Pt_GROUP_STRETCH_HORIZONTAL

Растягивает правую колонку виджетов при расширении группы



Pt_GROUP_STRETCH_FILL

Растягивает последний (ие) виджеты так, чтобы заполнить доступное свободное пространство в направлении, указанном в ориентации



☞ Не устанавливайте флаги **Pt_GROUP_EQUAL_SIZE_...** и **Pt_GROUP_STRETCH_...** для одного и того же направления – группа будет расширяться каждый раз, когда будет вычисляться его величина.

Для получения более полной информации см. описание виджета PtGroup в "Справочнике виджетов".

Расщепление группы на составляющие

Чтобы расщепить группу на отдельные виджеты:

1. Выберите группу
2. Сделайте одно из следующего:
 - Выберите пункт "Split Apart" из меню "Edit"
 - Нажмите <Ctrl>+<P>
 - Щёлкните на иконке "Split" на панели инструментов PhAB:



PhAB разберёт группу на составляющие виджеты и удалит контейнер PtGroup.

Управление привязкой с использованием анкеров – средств привязки

Вот общая ситуация компоновки, которая не управляется какой-бы то ни было политикой компоновки, рассматривавшейся нами. Предположим, что контейнер разделён на ряд панелей, ограниченных в своих размерах и местоположении. Обычно мы не желаем, чтобы панели перекрывались, и мы хотим управлять тем, как панели изменяют свои размеры, если сам контейнер увеличивается в размерах или сжимается. Механизм привязки обеспечивает это управление.

Анкеры предоставляются как механизм привязки позиции и размеров любого виджета внутри контейнера. Атрибут позиции и анкеры каждого потомка всегда используются для определения их позиций.

- ☞ В текущей версии microGUI Photon'a виджеты немедленно прикрепляются при создании. В более ранних версиях закрепление делалось, когда виджет реализовывался.

Анкер может быть задан для любой стороны порождённого виджета. Анкер прикрепляется к одной из сторон родителя. Он удерживает соответствующую сторону потомка на фиксированном расстоянии – смещении анкера от крепящей стороны родителя. Смещение анкера может быть также выражено как количественное соотношение ширины или высоты холста родителя.

Возможно – но не всегда желательно – прикреплять края виджетов за пределами холста его родителя.

Каждый раз при изменении размеров родителя позиция потомка (и возможно, его размеры) изменяются, чтобы сохранить это взаимоотношение. Если какая-либо сторона потомка не прикреплена к родителю, это позволяет ей свободно "плавать". Если Вы явно установили размеры и/или позицию для закреплённого виджета, смещения его анкера пересчитываются автоматически.

- ☞ При использовании PhAB Вы не задаёте смещения анкера. Вместо этого Вы позиционируете виджеты на желаемом смещении путём установки ресурсов позиции (Pt_ARG_POS) и размеров (Pt_ARG_DIM). PhAB вычисляет смещение анкера автоматически, основываясь на относительных размерах и позициях родителя и закреплённых потомков.

На ширину порождённого виджета влияют анкеры для его левой и правой сторон; на высоту – анкеры для верха и низа. Если любой из противоположных пар краёв позволено "плавать", стеснённость встречается при выборе только позиционировании виджета в соответствующих размерах. Это означает, что виджет может скользить в любом из четырёх направлений для удовлетворения анкерных ограничений. Если оба края "заякорены", виджет также должен изменять свои размеры в этом направлении.

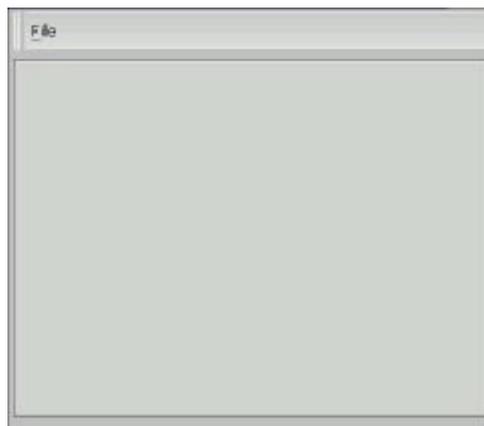


Рис. 7-1. Пример закрепления

- ☞ Если политика изменения размеров конфликтует с анкерами, флаги Pt_ARG_RESIZE_FLAGS переписывают Pt_ARG_ANCHOR_OFFSETS и Pt_ARG_ANCHOR_FLAGS.

Создание главного окна приложения представляет из себя простой пример использования анкерных ресурсов. В общем случае окно состоит по меньшей мере из двух частей: панель меню и рабочая область. Если рассматривать приложение, имеющее групповой виджет в рабочей области,

мы можем идентифицировать типы анкерov, необходимые для того, чтобы сделать изменение его размеров в ответ на изменение размеров виджета окна корректным. Каждый край рабочей области прикреплён к соответствующему краю окна. Смещение левого и верхнего анкерov установлены такими же, как атрибуты позиции для виджета. Они должны быть вычислены, чтобы разместить рабочую область под панелью меню. Размеры виджета установлены в соответствии с желаемым объёмом рабочего пространства.

При реализации окно размещает рабочую область там, где это задано в её атрибутах позиционирования. Размеры окна устанавливаются настолько большими, чтобы вместить рабочую область. Если окно изменяет размеры, ширина и высота рабочей области меняют свои размеры соответственно, поскольку все края закреплены. Если смещения анкера были определены корректно, позиция виджета не изменится.

Мы ничего не делаем с панелью меню, поскольку она автоматически прикрепляется к вершине и сторонам окна.

Ресурсы анкерov

Ресурс `Pt_ARG_ANCHOR_FLAGS` (определённый для `PtWidget`) управляет прикреплением анкерami. Среди анкерных флагов имеются три, связанные с каждым краем виджета, эти три флага позволяют каждому краю быть прикреплёнными одним из трёх возможных способов:

- прикрепиться к соответствующему краю его родителя
- прикрепиться к противоположному краю его родителя
- задать определённое местоположение, т.е. в какой-то пропорции от ширины или высоты виджета.

Эти флаги используют такие именуемые схемы:

`Pt_edge_ANCHORED_anchor`

где

edge является именем закрепляемого края и должно быть `TOP`, `LEFT`, `RIGHT` или `BOTTOM`, т.е. верхний, левый, правый и нижний края соответственно;

anchor является именем родительского края, к которому производится прикрепление, т.е. родительского *edge* (см. выше), или принимать значение `RELATIVE` для пропорционального анкера.

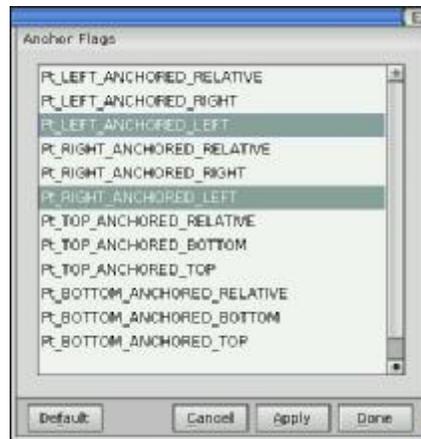
Таким образом, определены следующие флаги (не, ну уже для совсем непонятливых!!! Прим. пер.):

- `Pt_LEFT_ANCHORED_LEFT`
- `Pt_LEFT_ANCHORED_RELATIVE`
- `Pt_LEFT_ANCHORED_RIGHT`
- `Pt_RIGHT_ANCHORED_LEFT`
- `Pt_RIGHT_ANCHORED_RELATIVE`
- `Pt_RIGHT_ANCHORED_RIGHT`
- `Pt_TOP_ANCHORED_BOTTOM`
- `Pt_TOP_ANCHORED_RELATIVE`
- `Pt_TOP_ANCHORED_TOP`
- `Pt_BOTTOM_ANCHORED_BOTTOM`
- `Pt_BOTTOM_ANCHORED_RELATIVE`
- `Pt_BOTTOM_ANCHORED_TOP`

Пропорциональный анкер задаёт положение края как процентное отношение к одному из размеров родительского окна. Левый или правый края задаются как пропорция ширины родителя, и верхний или нижний края задаются как пропорция высоты родителя. Пропорция обеспечена смещением анкера и выражается в десятых долях процентов.

Установка флагов анкеров в PhAB

Чтобы установить флаги анкера, щёлкните на ресурсе флагов анкера (Pt_ARG_ANCHOR_FLAGS) и используйте редактор флагов PhAB:



Установка флагов анкеров в программном коде Вашего приложения

Вы можете также установить эти флаги из Вашего программного кода, используя метод, описанный в главе "Манипулирование ресурсами в программном коде приложения". Для удобства каждый набор флагов имеет объединённую маску битов:

- Pt_LEFT_IS_ANCHORED – выделяет биты, отвечающие за задание анкера для левого края
- Pt_RIGHT_IS_ANCHORED – выделяет биты, отвечающие за задание анкера для правого края
- Pt_TOP_IS_ANCHORED – выделяет биты, отвечающие за задание анкера для верхнего края
- Pt_BOTTOM_IS_ANCHORED – выделяет биты, отвечающие за задание анкера для нижнего края.

Таким образом, чтобы установить левый и правый края нашей панели меню в приведенном выше примере, необходимо инициализировать элемент списка аргументов следующим образом:

```
PtSetArg(&arg[n], Pt_ARG_ANCHOR_FLAGS,  
        Pt_LEFT_ANCHORED_LEFT | Pt_RIGHT_ANCHORED_RIGHT |  
        Pt_TOP_ANCHORED_TOP,  
        Pt_LEFT_IS_ANCHORED | Pt_RIGHT_IS_ANCHORED |  
        Pt_TOP_IS_ANCHORED);
```

При установке анкерных флагов из программного кода Вашего приложения все смещения анкера задаются путём использования ресурса Pt_ARG_ANCHOR_OFFSETS. Этот ресурс берёт структуру PtRect_t (см. "Справочник по библиотеке Photon") как значение. Верхний левый угол прямоугольника используется для определения смещения анкера для верхнего и левого краёв виджета, и нижний правый угол прямоугольника указывает смещение анкера для правого и нижнего краёв.

Так, например, чтобы сделать рабочую область в 90% от ширины окна с равным размером обеих сторон, левый и правый края прикрепляются, используя следующий код:

```
PhRect_t offsets;  
offsets.ul.x=50;  
offsets.lr.x=950;  
PtSetArg(&arg[n], Pt_ARG_ANCHOR_FLAGS,  
        Pt_LEFT_ANCHORED_RELATIVE | Pt_RIGHT_ANCHORED_RELATIVE,  
        Pt_LEFT_IS_ANCHORED | Pt_RIGHT_IS_ANCHORED);  
  
PtSetArg(&arg[n+1], Pt_ARG_ANCHOR_OFFSETS, &offsets, 0);
```

Помните, что пропорциональность, обеспечиваемая смещением анкера, выражается в десятых долях процента.

Установка ограничений по размерам или позиционированию без анкеров

Если Вы хотите поддерживать между позициями потомков более сложные связи относительно контейнера, или относительно друг друга, Вы должны захватывать события изменения размеров для контейнера. Виджетный класс `PtContainer` обеспечивает ответную реакцию `Pt_CB_RESIZE`, которую Вы можете использовать в этих целях.

Член `cbdata` структуры `PtCallbackInfo_t` (см. "Справочник виджетов Photon'a") является указателем на структуру `PtContainerCallback_t`, содержащую по меньшей мере следующие члены:

```
PtRect_t old_size.
```

Структура `Ptrect_t` (см. "Справочник библиотечных функций Photon"), определяющая предыдущий (старый) размер контейнера.

`PtRect_t new_size` Структура `PtRect_t`, определяющая новые размеры.

Глава 8. Генерирование, компилирование и запуск программного кода на исполнение

В этой главе описывается, как сгенерировать, сформировать, скомпилировать и запустить на исполнение программный код для приложения PhAB:

- Использование диалога "Build+Run"
- Генерирование программного кода приложения
- Как организовать файлы приложения
- Редактирование исходного кода
- Компилирование и линковка
- Запуск приложения на исполнение
- Отладка
- Включение в Ваше приложение не-PhAB файлов
- Создание DLL из приложения PhAB

PhAB автоматически генерирует все, что требуется для того, чтобы превратить Ваше приложение в работающий исполняемый файл, в том числе:

- кодирует ту часть Вашего приложения, которая обрабатывает пользовательский интерфейс
- создаёт заготовки файлов на C и/или C++ для специфических для данного приложения ответных реакций, функций установки модулей, функций инициализации приложения, и прочая
- генерирует все файлы, требующиеся для компилирования и линковки приложения – файл Makefile, глобальный хедер, главный файл и файл-прототип.

Выполняя всё это, PhAB позволяет Вам преуспеть в работе по написанию кода, обеспечивающего основную функциональность Вашего приложения.

В большинстве случаев генерирования программного кода Вы можете использовать диалог "Build+Run" или пункт "Generate" меню "Application". Однако Вы можете также генерировать некие файлы-заглушки на C и C++ в тех затруднительных случаях, когда при обустройстве Вашего приложения используются различные диалоги; используйте вот такие иконки, располагающиеся следом за областью имени функции или файла:



Это означает, что Вы вольны редактировать функцию ответной реакции, пока она ещё в процессе прикрепления её к виджету. Вы не можете перейти в диалог "Build+Run", сгенерировать оттуда код, а затем вернуться назад, чтобы написать функцию.

Использование диалога Build+Run

Рассматривайте диалог "Build+Run" как центр разработки для создания Вашего приложения. Из этого диалога Вы можете:

- сгенерировать код приложения
- собрать (скомпилировать и слинковать) Ваше приложение
- отладить Ваше приложение
- отредактировать код
- запустить Ваше приложение на исполнение

Чтобы открыть диалог "Build+Run", выберите пункт "Build+Run" из меню "Application" или нажмите <F5>.

☞ Если пункт "Build+Run" тусклый, это значит, что Вы ещё не дали имя Вашему приложению. Для получения более полной информации см. раздел "Сохранение приложения" в главе "Работа с приложениями".

PhAB автоматически сохраняет Ваше приложение, когда Вы открываете диалог "Build+Run".

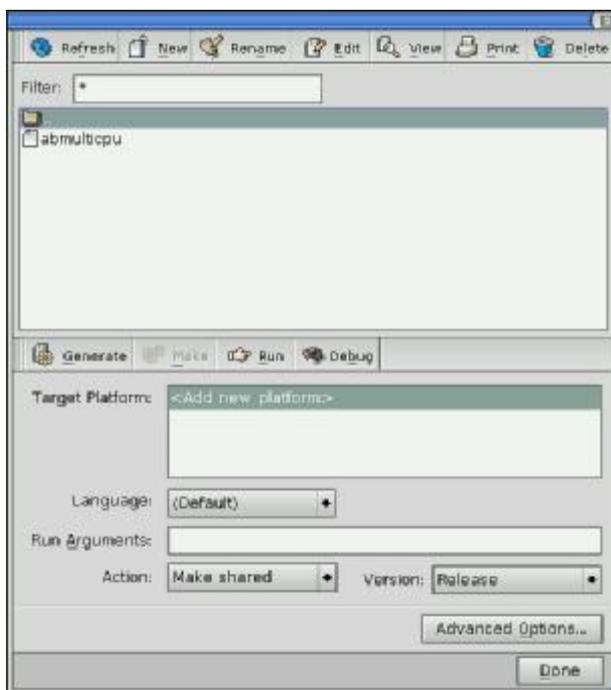


Рис. 8-1. Простой сеанс "Build+Run"

Прокручивающийся список отображает файлы с исходным кодом приложения, которые генерирует PhAB, а также все те файлы, что созданы Вами "вручную". Этот список может быть пустым, если Вы проектируете новое приложение и ещё не сгенерировали никакого кода.

Генерирование программного кода приложения

Когда Вы делаете изменения в Вашем приложении, даже внутри Ваших собственных исходных файлов, Вам необходимо сгенерировать код приложения. Выполнение этого обеспечивает, что будет обновлён хедер-файл прототипа proto.h. Вы можете смело генерировать код в любой момент – PhAB не будет переписывать какой-либо код, который Вы добавляли в заглушки, сгенерированный ранее. Перед генерацией кода PhAB сохранит Ваше приложение, если Вы модифицировали какие-либо модули. Чтобы минимизировать время компиляции, PhAB компилирует только те файлы, которые изменялись.

☞ Если Вы планируете использовать в Вашем приложении глобальный хедер, Вы должны установить хедер перед тем, как генерировать какой-либо код. Для получения более подробной информации см. раздел "Задание установочной информации приложения" в главе "Работа с приложениями" и раздел "Глобальный хедер-файл" в главе "Работа с программным кодом".

Чтобы сгенерировать программный код Вашего приложения:

1. Щёлкните на кнопке "Generate" диалога "Build+Run" или выберите пункт "Generate" из меню "Application".
2. Если Вы ещё не выбрали платформу (т.е. комбинацию компилятора и процессора), появится диалог:

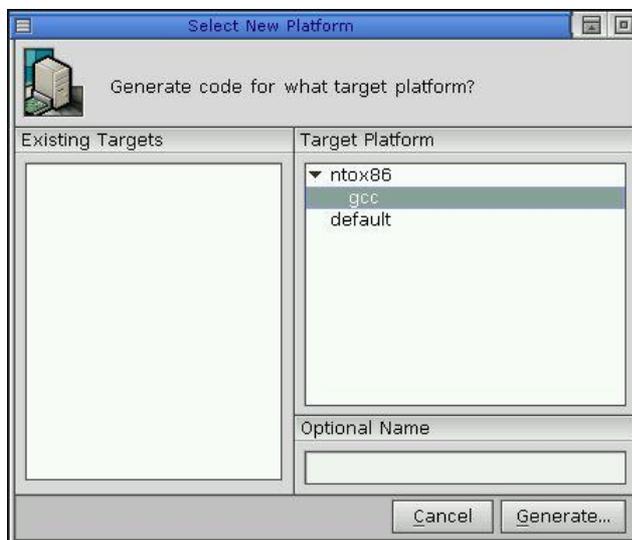


Рис. 8-2. Селектор платформы

3. После того как Вы выбрали платформу, дождитесь, чтобы счётчик прогресса достиг 100%.
4. Щёлкните на "Done" в диалоге прогресса.

Если Вы сгенерировали код в диалоге "Build+Run", список файлов обновляется, показывая все сгенерированные файлы с программным кодом.

Что генерирует PhAB

PhAB генерирует различные файлы и сохраняет их в директории приложения src.



Любое имя файла, начинающееся с префикса "ab", является файлом PhAB и не должно никогда модифицироваться. Если Вы попытаетесь отредактировать ab-файл, Вы можете потерять результаты работы (когда PhAB переписет файл) или получить некорректное поведение (когда файлы выйдут из синхронизма).

Вы можете модифицировать любые другие файлы, генерируемые PhAB, при некоторых условиях. Эти условия описаны в следующих разделах.

Вот файлы, генерируемые PhAB:

Makefile	Используется для компилирования и линковки приложения
Usemsg	Сообщение пользователя для приложения
abHfiles	
abOfiles	
abSfiles	Внешние ссылки PhAB в Makefile
abdefine.h	Содержит все сгенерированные PhAB'ом декларации. PhAB включает этот хедер во все файлы C
abevents.h	Содержит все ответные реакции приложения.
abimport.h	Хедер внешней ссылки, включаемый во все файлы C. См. раздел "Прототипы функций" ниже
ablinks.h	Содержит все определения модулей приложения

abmain.c	Главный C-файл приложения. Этот файл начинается с "ab", так что не модифицируйте его
abmain.cc	Если PhAB определяет какие-либо функции C++ в Вашем приложении, он генерирует abmain.cc вместо abmain.c. Этот файл также начинается с "ab", так что не модифицируйте его
abplatform	Содержит список директорий платформ для приложения
abvars.h	Содержит все сгенерированные PhAB'ом глобальные переменные
abwidgets.h	Содержит все списки данных PhAB
proto.h	Содержит прототипы приложения – см. раздел "Прототипы функций" ниже. Не переименовывайте этот файл

Управление версиями

Вот файлы, которые Вам надо сохранять, если Вы используете ПО управления версиями (PhAB может генерировать некоторые из них, но хранить их все – не очень хорошая идея):

abapp.dfn	Ответные реакции и другая информация – это бинарный файл.
wgt/*	Ресурсы виджетов – они могут выглядеть похожими на текстовые файлы, но это бинарники.
src/*.c,cc,cpp,C,h	Файлы с исходным кодом и хеадеры.
src/*files	Файлы, относящиеся к не-PhAB исходным файлам. Убедитесь, что сохраняете также не-PhAB исходники.
src/Makefile,	
src/*/Makefile	Все make-файлы.
application_name.ldb	Языковая база данных Вашего приложения. Сохраните также какие-либо файлы перевода.

Вам понадобится содержать сопоставляемый набор всех файлов, генерируемых PhAB; сохраните ту же самую версию файлов abapp.dfn, src/ab* и wgt/*.wgt?

Некоторые рекомендации по использованию CVS

Сохранять приложение в PhAB легче, чем в RCS (RCS – Revision Control System; CVS – вероятно, Control Version System – система управления версиями. Прим. пер.).

Вот несколько вещей, которые надо помнить:

- Пометьте файлы *.wgt? и abapp.dfn как бинарные (-kb).
- Поскольку бинарные файлы не могут быть объединены, попытайтесь исключить возможность модификации бинарных файлов несколькими людьми одновременно. CVS не поддерживает запирация; самое точное, что Вы можете получить, это установить "watch" (наблюдение) в abapp.dfn (cvs отслеживает abapp.dfn).

При таком подходе, если Вы заканчиваете работу над приложением, Ваша копия файла abapp.dfn имеет атрибут "только для чтения" и PhAB не позволит Вам загрузить приложение. Если Вы хотите модифицировать приложение, Вы запускаете редактирование файла abapp.dfn с помощью CVS, которая делает файл доступным для чтения. Хотя это и не препятствует другому народу выполнить эти же действия, по крайней мере добавляет Вас в список "редакторов" сервера CVS, который другие пользователи могут опрашивать.

Прототипы функций

PhAB генерирует прототипы функций, используемые для компиляции, чтобы проверить, правильно ли вызываются Ваши функции. Эти прототипы располагаются в abimport.h и, возможно, в proto.h. Вот сравнение этих файлов:

proto.h	abimport.h
Генерируется при синтаксическом разборе Вашего кода	Генерируется при просмотре установок Вашего приложения
Генерируются прототипы для всех функций	Генерируются только прототипы, известные в PhAB (ответные реакции, установочные функции, ресурсы указателей на функции)
Вы можете иметь проблемы с препроцессорными директивами (см. раздел "Потенциальные проблемы при генерировании файла proto.h), несвойственными языку C конструкциями, синтаксическими ошибками и кодом C++	Прототипы не зависят от исходного кода
Не работает с C++	Содержит predefined #ifdefs и расширенные "C" декларации, требуемые для C++.
Прототипы совпадают с тем, как выглядят функции	Прототипы совпадают с тем, как предположительно выглядят функции – если исходный код различается, компилятор сможет это определить.

Чтобы подавить генерирование прототипов в proto.h:

1. Нажмите <F2> или выберите в меню "Application" пункт "Startup Info/Modules", чтобы открыть диалог "Application Startup Information".
2. Щёлкните на кнопке "Generate empty "proto.h" file".

Потенциальные проблемы с генерированием proto.h

Для повышения скорости программа, сканирующая Ваши файлы с исходным кодом на предмет прототипов функций, игнорирует препроцессорные директивы. Это может привести к определённым проблемам в файле proto.h.

Например, пусть мы имеем следующий код:

```
#ifdef DOUBLE
    for (i=0; i<18; i++, i++) {
#else
    for (i=0; i<18; i++) {
#endif
    x += 2*(i+x);
    y += x
}
```

Поскольку процессорные директивы игнорируются, генератор прототипа видит следующее:

```
for (i=0; i<18; i++, i++) {
for (i=0; i<18; i++) {
    x += 2*(i+x);
    y += x
}
```

Две открытые фигурные скобки вызовут определённое замешательство, и будет сгенерирован неверный прототип. Проверяйте подобные вещи, если генератор прототипов создаёт неверные прототипы.

Чтобы исправить код, приведённый выше, мы должны удалить открывающие скобки и разместить их в строке после #endif. Или же мы можем сделать таким образом:

```
#ifdef DOUBLE
#define ADDAMOUNT 2
#else
#define ADDAMOUNT 1
#endif
```

```
for (i=0; i<18; i += ADDAMOUNT) {  
    x += 2*(i+x);  
    y += x;  
}
```

Как организовать файлы приложения

PhAB хранит каждое приложение как структуру директорий. Эта структура состоит из главной директории, хранящей файл описания приложения, двух поддиректорий, содержащих файлы модулей и исходный программный код приложения и, потенциально, директории для различных платформ разработки:

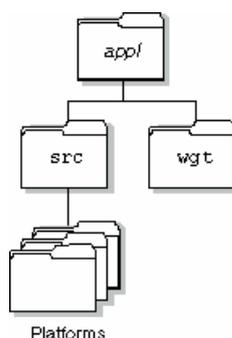


Рис. 8-3. Директории для приложения PhAB

Если Вы первый раз сгенерировали Ваше приложение с ранней версией Photon'a, оно могло быть создано как одноплатформенное приложение. В этом случае размещение файлов слегка различается, как описано в последующем разделе.

Вы можете выбрать платформы, для которых компилируется Ваше приложение. Если Вам не требуется чего-то иного, Вы можете выбрать в качестве платформы принимаемое по умолчанию (выбрать "default"). Если Вы выбрали default, при инсталляции новых версий компилятора они будут использоваться автоматически.

Многopлатформенные приложения

Вот что содержит каждая директория многоплатформенного приложения:

app/

Имя этой платформы – то же, что имя Вашего приложения. Она содержит файл описания приложения `abapp.dfn`. Поскольку этот файл является принадлежностью PhAB, Вам не следует пытаться его модифицировать.

app/src

Директория `src` содержит исходный программный код, хедеры и файл `Makefile`, сгенерированные PhAB, а также любой код, созданный Вами собственноручно. Диалог "Build+Run" при раскрытии отображает содержание этой директории. Для получения более подробной информации о файлах, содержащихся в этой директории, см. раздел "Что генерирует PhAB" в этой главе.

app/src/platforms

Эти директории содержат файл `Makefile`, объектные файлы и исполняемые файлы для выбранной платформы. Файл `Makefile` в директории `src` запускает их в платформенных директориях.

app/wgt

Директория `wgt` содержит модули Вашего приложения. Поскольку каждый тип модуля имеет своё отличающееся расширение файла, довольно легко опознать соответствующий

нужный Вам модуль при импортировании модулей из другого приложения. Для получения более полной информации см. подручные таблицы в разделе "Типы модулей" главы "Работа с модулями".



Всегда используйте PhAB, чтобы отредактировать файлы модулей в директории wgt. Не пытайтесь редактировать эти бинарные файлы другим редактором. Никогда не модифицируйте какие-либо файлы, начинающиеся с "ab".

Одноплатформенные приложения

Вот что содержит каждая директория одноплатформенного приложения:

appl

Имя этой директории – то же, что имя Вашего приложения. Она содержит файл описания приложения abdefn.app. Поскольку этот файл является собственностью PhAB, Вам не следует пытаться его модифицировать. После того как Вы откомпилировали и слинковали приложение, в этой директории также размещается исполняемый файл.

appl/src

Директория src содержит файлы с исходным кодом хедера, объектные файлы и файл Makefile, сгенерированные PhAB, а также любые файлы с исходным кодом, созданные Вами собственноручно. При открытии диалог "Build+Run" отображает содержание этой директории.

Для получения более полной информации о файлах, хранящихся в этой директории, см. раздел "Что генерирует PhAB" в этой главе.

appl/wgt

Директория wgt содержит модули ВАшего приложения. Поскольку каждый тип модуля имеет своё отличающееся расширение файла, довольно легко опознать соответствующий нужный Вам модуль при импортировании модулей из другого приложения.

Для получения более полной информации см. подручные таблицы в разделе "Типы модулей" в главе "Работа с модулями".



Всегда используйте PhAB, чтобы редактировать файлы модулей в директории wgt. Не пытайтесь редактировать эти бинарные файлы другим редактором. Никогда не модифицируйте какие-либо файлы, начинающиеся с "ab".

Преобразование к мультиплатформенности

Если у Вас имеется одноплатформенное приложение, построенное в одной из ранних версий Photon'a, Вы можете преобразовать его под мультиплатформенность, но это необязательно. PhAB работает с обоими типами приложения.

Чтобы преобразовать к мультиплатформенности, выберите пункт "Convert to Multiplatform" из меню "Application". PhAB переместит все существующие файлы Makefile в src/default/Makefile.old. Используйте пункт "Generate" в меню "Application", или команду "Generate" в диалоге "Build+Run", чтобы сгенерировать новый файл Makefile для нужных платформ, и затем отредактируйте их, чтобы передать какие-либо требующиеся изменения из старого файла Makefile в новый.

Редактирование исходного кода

Как только Вы сгенерировали программный код Вашего приложения, Вы увидите модули с исходным программным кодом C и/или C++, отображаемые в списке файлов диалога "Build+Run".

Рядом со списком файлов Вы увидите несколько кнопок, позволяющих выполнить над файлами различные действия. Чтобы редактировать, просматривать или удалять исходный код:

1. Щёлкните на файле с исходным кодом
2. Щёлкните на соответствующей кнопке действия ("Edit" – редактирование, "View" – просмотр, "Delete" – удаление, ...).

Вы можете также начать редактирование файла, дважды щёлкнув на его имени.

Выбор редактора или броузера

Чтобы выбрать, какой редактор или броузер вызывать по кнопкам "Edit" или "View", см. раздел "Подгонка Вашего окружения PhAB" в главе "Окружение PhAB".

Создание модуля с исходным кодом

Чтобы создать новый модуль с исходным кодом:

1. Щёлкните на кнопке "Create", чтобы открыть диалог "Create File", затем наберите имя нового файла.
2. Если Вы хотите создавать файл, используя шаблон (для хедер-файла или ответной реакции виджета), выберите формат из комбинированного элемента управления "Template".
3. Щёлкните на "Create". Вы увидите окно терминала, отображающего либо пустой файл, либо файл, содержащий шаблон.

Если Вы создали какие-либо файлы, щёлкните на кнопке "Refresh" для просмотра директории приложения и обновления списка файлов в левой части диалога "Build+Run".

Изменение отображения файлов

Для управления тем, какие файлы отображаются диалогом "Build+Run", используйте следующее:

- "Refresh" – Вынуждает пересмотреть директорию исходных кодов приложения для обеспечения правильности Вашего списка файлов.
- File Spec – позволяет Вам задать шаблон имён отображаемых файлов.

Компилирование и линковка

После генерации кода приложения Вам необходимо:

1. Выбрать библиотеки, используемые Вашим приложением
2. Использовать команду make для компиляции и линковки Вашего приложения.

Выбор библиотек

PhAB позволяет Вам использовать с Вашим приложением следующие библиотеки:

- Статические библиотеки – компоновка библиотек PhAB и Photon в исполняемый файл приложения. Приложение получается больше по размеру, нежели при использовании совместно используемой библиотеки, но выполняется без библиотек совместного использования. Это может быть полезным во встраиваемых приложениях.
- Совместно используемые библиотеки – библиотеки не включаются в приложение. Приложение получается намного меньшим по размеру, но для своего выполнения требует совместно используемые библиотеки Photon'a.

По умолчанию установлено использование совместно используемых библиотек. При запуске PhAB Вы также можете задать список библиотечных функций ответных реакций.

Для получения более полной информации см. описание appbuilder в "Справочнике утилит QNX".

Запуск команды make

Как только Вы выбрали тип библиотеки, Вы готовы компилировать и линковать. Когда Вы впервые генерируете Ваше приложение, PhAB создаёт файл Makefile в директории src (плюс по файлу Makefile для каждой платформы, выбранной при мультиплатформенной разработке), так что Вы можете собирать Ваше приложение. Последующие генерации кода непосредственно не модифицируют файл – вместо этого они обновляют внешние файлы, на которые ссылается Makefile. После того как Makefile сгенерирован, Вы вольны его модифицировать, при нескольких условиях:

- PhAB размещает в Makefile ссылки на внешние файлы для объектных файлов, файлов с исходным кодом и хедер-файлов, как сгенерированных им, так и созданных пользователем. Не удаляйте эти ссылки.
- PhAB также использует три ссылки на целевые имена, называемые app, shr и proto. Не переименовывайте эти целевые объекты.

Целевые объекты app и shr используются для компилирования и линковки приложения со статическими или совместно используемыми библиотеками. Целевой объект proto используется для генерирования файла прототипа приложения proto.h; см. раздел "Генерирование прототипов функций" ниже в этой главе.

По умолчанию файл Makefile совместим с установленной командой "make". Вы можете преобразовать файл в формат, соответствующий предпочитаемой Вами команде "make" – просто убедитесь, что метод ссылки на внешние файлы ещё совместим.

Для получения более полной информации см. раздел "Включение неPhAB-шных файлов в Ваше приложение" ниже в этой главе.

Чтобы собрать Ваше приложение:

1. Щёлкните на кнопке "Make", чтобы открыть диалог "Make Application", и запустите на исполнение make.
2. Если во время исполнения make будут обнаружены какие-либо ошибки или предупреждения, PhAB сделает доступными кнопки "Edit" и "Restart".
Чтобы редактировать первый файл, содержащий ошибки, щёлкните на "Edit". После решения проблем щёлкните на "Restart", чтобы запустить make снова. Чтобы остановить в любой момент make, щёлкните на "Abort".
3. После того как приложение откомпилировано и слинковано, PhAB делает доступной кнопку "Done" диалога "Make". Щёлкните на "Done", чтобы закрыть диалог. Кнопка "Done" становится также доступной, если щёлкнуть на "Abort".

Модифицирование команды make

По умолчанию PhAB использует установленную make-команду, чтобы собрать Ваше приложение. Если Вам необходимо каким-либо образом изменить эту команду, щёлкните на кнопке "Build Preferences".

- ☞ Все изменения, сделанные Вами в установках "Build Preferences", сохраняются не как глобальные установки, а вместе с самим приложением.

Запуск приложения на исполнение

Как только Ваше приложение без ошибок откомпилировано и слинковано, оно готово к исполнению. Просто следуйте этим шагам:

1. Если Вы использовали PhAB для создания многоязыкового приложения, Вы можете перед запуском Вашего приложения на исполнение выбрать язык в диалоге "Build+Run". Для получения более полной информации см. главу "Поддержка международных языков".
2. Если Ваше приложение требует аргументы командной строки, введите их в поле "Run Arguments".
3. Щёлкните на кнопке "Run Appl".

☞ Когда Ваше приложение исполняется, его рабочей директорией является та, что отображена в диалоге "Build+Run".

Если Вы используете в Вашем приложении такие функции как printf(), то если Вы запустили приложение из PhAB, вывод идёт на Вашу консоль. Чтобы увидеть этот вывод:

- Откройте окно pterm и используйте утилиту ditto для просмотра консоли (ditto описана в "Справочнике утилит QNX")
или
- Откройте pterm и запустите приложение не из PhAB, а из pterm.

PhAB остаётся активным, пока исполняется Ваше приложение. Чтобы между ними переключаться, используйте панель задач Window Manager'a.

Отладка

PhAB позволяет Вам запускать Ваше приложение под отладчиком, который может оказаться полезным, если Ваше приложение терпит крах или ведёт себя неверно.

☞ Чтобы запустить Ваше приложение из отладчика, щёлкните на кнопке "Debug Application". Этот отладчик запускается в терминальном окне. Ваше приложение отображается при запуске его из отладчика.

Чтобы переключиться между отладчиком и приложением, используйте панель задач Window Manager'a.

Модифицирование команды отладчика

Принимаемым по умолчанию отладчиком является gdb. Если Вам необходимо каким-либо образом изменить эту команду, щёлкните на кнопке "Advanced Options" и отредактируйте команду отладчика. Если Вы используете вызовы printf() для отладки Вашей программы, простейший способ увидеть вывод – это изменить принимаемый по умолчанию отладчик на:

```
pterm -z
```

Когда Вы щёлкаете на кнопке "Debug Application" в диалоге "Build+Run", PhAB создаёт pterm, запускающий Ваше приложение. Вывод программы появляется в окне pterm. Опция -z оставляет окно pterm открытым вплоть до явного закрытия. Для получения более полной информации по pterm см. "Справочник утилит QNX6". Вы можете даже использовать printf() и gdb вместе, установив принимаемый по умолчанию отладчик в:

```
pterm gdb
```

Когда Вы щёлкните на кнопке "Debug Application" в диалоге "Build+Run", PhAB запускает pterm, который запускает gdb, запускающий Ваше приложение. Вы можете затем использовать gdb и видеть печатаемый программой вывод.

- ☞ Все изменения, сделанные Вами в установках "Build Preferences", сохраняются не как глобальные установки, а вместе с самим приложением.

Включение в Ваше приложение не-PhAB файлов

Ваше приложение может включать файлы, которые были созданы не в PhAB, но Вам необходимо сообщить PhAB'у, как их отыскать.

Мультиплатформенные приложения

PhAB генерирует пустой список следующих файлов в директории src, и Вы можете его редактировать:

indHfiles	Не PhAB-шные хедер-файлы. Например: MYHDR=../header1.h ../header2.h
indOfiles	Не PhAB-шные файлы объектных модулей. Например: MYOBJ=file1.o file2.o
indSfiles	Не PhAB-шные файлы исходного кода. Например: MYSRC=../file1.c ../file2.c

- ☞ Помните о необходимости задания файловых имён относительно местонахождения файлов Makefile. Для мультиплатформенного приложения они являются относительными к директории платформы:
 - хедер-файлы и файлы исходного кода обычно располагаются в родительской директории src, так что их имя начинается с ../
 - объектные модули обычно располагаются в той же директории, что и файлы Makefile

Одноплатформенные приложения

Одноплатформенные приложения из более ранних версий Photon не имеют файлов indHfiles, indOfiles и indSfiles. Вместо этого Вы находите MYHDR, MYOBJ и MYSRC в Вашем файле Makefile и можете там задавать имена файлов.

- ☞ Помните о необходимости задания файловых имён относительно местонахождения файла Makefile. Для одноплатформенного приложения они являются относительными к директории src.

Добавление библиотек

Если Ваше приложение использует библиотеку, которая по умолчанию не включена в Makefile, Вы можете её добавить, отредактировав следующие переменные:

- LDFLAGS – используется при линковке вместо статических библиотек Photon'a.
- SDFLAGS – используется при линковке вместо библиотек совместного доступа Photon'a.

Создание результирующей DLL как приложения PhAB

Вы можете создать приложение PhAB как DLL, однако не существует опции PhAB, позволяющей Вам это сделать. PhAB ничего не знает о создании DLLей; существует библиотека PhAB, позволяющая Вам преобразовать приложение PhAB в DLL.

- ☞ Приложение, загружающее Вашу DLL, должно быть приложением PhAB, так что, по существу, библиотека PhAB инициализирована. Даже несмотря на то, что PhAB позволяет Вам при старте устанавливать функцию инициализации и окна открытия, они игнорируются, когда Ваше приложение является DLL. Это происходит потому, что обычно запуск выполняется путём вызова функции main(), а в DLL функция main() не вызывается. Не пытайтесь также вызывать её из своего собственного кода.

Компилирование и линковка

Вообще Вы можете преобразовать любое приложение (как создание PhAB'ом, так и нет) в DLL, добавив `-shared` к флагам компилятора и линковщика (и вероятнее всего добавляя расширение `so` или `dll` к имени файла). Вам необходимо также установить для линкера опцию `-Bsymbolic`, чтобы быть уверенным, что локально определённые символы, используемые DLL, не будут переписаны другими символами с такими же именами из исполняемого файла.

Чтобы выполнить эти изменения для приложения PhAB, следует сделать в Makefile следующее:

- Добавить `-shared` к CFLAGS
- Добавить `-shared -Wl, -Bsymbolic` к LDFLAGS и SDFLAGS.
- Найти все случаи появления исполняемого имени и добавить к нему расширение `so` или `dll`.

Инициализация DLL

Обычно DLL определяет функцию инициализации, которую вызывает приложение после того, как оно вызвало функцию `dlopen()` для загрузки DLL. Функция инициализации Вашей DLL требует полный путь к DLL.

Перед вызовом какого-либо кода PhAB функция инициализации должна вызвать `ApAddContext()` подобным образом:

```
ApAddContext(&AbContext, fullpath);
```

Аргументами здесь являются:

- AbContext – глобальная структура данных, которую PhAB перемещает в `abmain.c`
 - ☞ Эта структура имеет одно и то же имя во всех приложениях PhAB или DLL, так что Вы должны линковать Вашу DLL с опцией `-Bsymbolic`, как упомянуто выше.
- fullpath – полный путь к DLL, подходящий для передачи в функцию `open()`.

Вы можете вызвать функцию `ApAddContext()` более одного раза, но Вы должны отслеживать, как много раз Вы её вызывали.

`ApAddContext()` возвращает 0 в случае успешного выполнения или `-1` при неудаче. Не выполняйте вызов какой бы то ни было функции `Ap*`, если вызов функции `ApAddContext()` завершился неудачей.

Выгрузка Вашей DLL

Когда приложение готово выгрузить DLL, оно обычно вызывает в DLL функцию очистки. В функции очистки Вашей DLL Вы должны:

- Закрыть все базы данных виджетов, открывавшиеся Вашей DLL
- Уничтожить все виджеты PhAB, принадлежащие Вашей DLL
- Если Ваша DLL определила классы виджетов, вызвав функцию `ApAddClass()`, удалить их, вызвав функцию `ApRemoveClass()`

- Вызвать функцию `ApRemoveContext()` подобным образом:

```
ApRemoveContext (&AbContext);
```

Вы должны вызвать `ApRemoveContext()` столько раз, сколько раз Вы успешно вызывали `ApAddContext()`. После того как Вы вызвали `ApRemoveContext()`, Ваша DLL не должна вызывать какие-либо функции PhAB.

Глава 9. Работа с программным кодом

PhAB делает простым создание пользовательского интерфейса с приложением. Как только PhAB сгенерировал заготовки программного кода для Вашего приложения, Вам необходимо написать ту часть, которая делает приложение что-то выполняющим. Эта глава описывает, как работать с программным кодом для приложения PhAB.

Она включает следующие разделы:

- Переменные и декларации
- Глобальный хедер-файл
- Имена функций и файлов
- Функция инициализации
- Установочные функции модулей
- Функции ответных реакций кодового типа
- Типы данных о геометрии
- Таймеры
- Меню инициализации
- Задержка обновления и принудительное обновление экрана

☞ Для получения информации об использовании в программе Photon нитей см. раздел "Параллельные операции".

Переменные и декларации

Переменные и декларации виджетов

PhAB создаёт глобальные переменные и декларации для каждого созданного Вами модуля, и каждого виджета, имеющего уникальное имя экземпляра. Это делает простым доступ к виджетам из Вашего программного кода приложения.

Глобальная переменная представляет имя виджета и определена в файле `abvars.h`. Каждая глобальная переменная принимает такую форму:

- `ABN_widget_name` – где `widget_name` является именем виджета или именем экземпляра модуля, который Вы определили в панелях управления ресурсами или ответными реакциями. Значение этой переменной является уникальным в пределах всего приложения.

Декларация представляет указатель на экземпляр виджета и определена в файле `abdefine.h`. Этот файл, который включается во все файлы с кодом на языке C Вашего приложения, также определяет внешнюю ссылку на глобальные переменные. Каждая декларация принимает такую форму:

- `ABW_widget_name` – где `widget_name` является именем виджета или именем экземпляра модуля, который Вы определили в панелях управления ресурсами или ответными реакциями.

☞ PhAB не создаёт декларации `ABW_...` для модулей меню или пунктов меню. Обычно меню не живут достаточно долго, так что декларации для них не очень-то полезны. Если Вам необходимо изменять ресурсы `PtMenu`, создайте установочную функцию для модуля меню и делайте работу там. См. раздел "Установочные функции модулей" ниже.

Когда PhAB обнаруживает уникальное имя экземпляра, он генерирует имя глобальной переменной и декларацию виджета. Например, если Вы изменили имя экземпляра виджета класса `PtButton` – на "done", PhAB сгенерирует следующее:

- `ABN_done`
- `ABW_done`

- ☞ Глобальная переменная и декларация виджета будут иметь силу только после того как виджет будет создан, и до тех пор, пока не будет уничтожен.

Использование глобальной переменной и декларация виджета

Давайте теперь посмотрим несколько примеров того, как Вы можете использовать глобальное имя и декларацию виджета из программного кода приложения. Во-первых, вот пример использования глобальной переменной `ABN_done` и функции `ApName()` для проверки конкретного виджета в ответной реакции:

```
int mycallback(PtWidget_t * widget, ...) {
    /* проверка конкретного виджета */
    if (ApName(widget) == ABN_done) {
        /* выполнение обработки кнопки */
    }
    return(Pt_CONTINUE);
}
```

Следующий пример использует `ABW_done`, чтобы изменить цвет фона виджета `done` на красный (для получения более полной информации см. главу "Управление ресурсами в программном коде приложения"):

```
int mycallback(PtWidget_t * widget, ...) {
    PtSetResource(ABW_done, Pt_ARG_FILL_COLOR, Pg_RED, 0);
    return(Pt_CONTINUE);
}
```

- ☞ Помните, что глобальная переменная и декларация виджета будут иметь силу только после того, как виджет будет создан, и до тех пор, пока не будет уничтожен.

Обработка множества экземпляров окна

Если у Вас имеется более чем один экземпляр модуля окна, отображаемых одновременно, то у Вас будут проблемы, связанные с получением доступа к виджетам в окне. Проблема произрастает из того факта, что `ABW_instance_name` для любого виджета в модуле окна указывает на последний созданный экземпляр этого виджета. Если у Вас имеется более одного экземпляра окна, то у Вас имеется и более одного созданного экземпляра виджетов внутри окна.

Пусть, скажем, Вы имеете следующий модуль окна:



Рис. 9-1. Простое окно поиска

Примем, что имена экземпляров следующие:

- `search_win` для окна
- `name_txt` для текстовой области
- `search_btn` для кнопки

Если у Вас на экране одновременно имеется два экземпляра окна и пользователь щёлкнул на кнопке "Search", как Вы получите значение в текстовом виджете "Name"? Поскольку имеются два экземпляра окна, существует и два экземпляра текстового виджета. `ABW_name_txt` указывает на последний созданный экземпляр текстового виджета.

Решение основано на том факте, что для ссылки на оба экземпляра виджета `name_txt` может использоваться `ABN_name_txt`, обеспечивающий Вас указателем на виджет в том окне, которое содержит нужный текстовый виджет. Это осуществляется использованием функции `ApGetWidgetPtr()`:

```
PtWidget_t *window_wgt, *text_wgt;
text_wgt = ApGetWidgetPtr(window_wgt, ABN_name_txt);
```

Где Вы получите `window_wgt`? В вышеприведенном случае Вы имеете ответную реакцию кодового типа на кнопке "Search". Первым параметром, передаваемым кодовой ответной реакцией, является указатель на виджет кнопки "Search". Для получения указателя на окно, содержащее кнопку "Search", Вы можете использовать функцию `ApGetInstance()`. Таким образом ответная реакция станет такой:

```
int search_callback(PtWidget_t .widget, ApInfo_t .apinfo, PtCallbackInfo_t .cbinfo) {
char .name;
PtWidget_t .window_wgt, .txt_wgt;
/* Получение окна, в котором находится кнопка "Search" */
window_wgt = ApGetInstance(widget);
/* В данном окне ищем текстовый виджет */
text_wgt = ApGetWidgetPtr(window_wgt, ABN_name_txt);
/* Теперь получим текст */
PtGetResource(text_wgt, Pt_ARG_TEXT_STRING, &name, 0);
/* Переменная 'name' теперь указывает на корректный текст. */
/* Обработка текста соответствующим образом */
...
return (Pt_CONTINUE);
}
```

Декларация внутренних связей

PhAB генерирует декларацию для каждой внутренней связи, определённой в Вашем приложении:

- `ABM_internal_link_name` – где `internal_link_name` является указателем на внутреннее определение модуля.

Для получения более полной информации об использовании внутренних связей см. главу "Доступ к модулям PhAB из программного кода".

Декларации иконок

PhAB также предусматривает для доступа к иконкам приложения две декларации. Имена соответствуют принимаемым по умолчанию, которое Вы всегда можете использовать, когда определяете иконки:

- `ABW_LIcon` – указатель на экземпляр крупной иконки. Эта иконка используется только при использовании на полке иконок. Для получения более полной информации см. раздел "Модули иконок" в главе "Работа с модулями".
- `ABW_SIcon` – указатель на экземпляр мелкой иконки, отображающейся на панели задач. Поскольку Вы можете использовать этот указатель для доступа к иконке, легко осуществить для иконки требуемую переделку или анимацию.

Глобальный хедер-файл

PhAB позволяет Вам определить один глобальный хедер-файл для каждого приложения. PhAB генерирует этот файл только однажды, в первый раз, когда Вы генерируете код приложения.

☞ Как только Вы определили хедер, PhAB автоматически включает его во все генерируемые заготовки C и C++ файлов. Таким образом, лучше всего определить хедер, когда Вы впервые создаёте приложение. См. раздел "Установка стартовой информации приложения" в главе "Работа с приложениями". Вы можете модифицировать хедер-файл в любой нужный Вам момент.

Вот удобный способ использования этого одиночного хедер-файла для одновременного определения всех Ваших глобальных переменных и внешних ссылок на эти переменные:

```
/* Хедер "globals.h" для приложения my_appl */  
  
#include <Pt.h>  
  
#ifdef DEFINE_GLOBALS  
  
#define GLOBAL  
#define INIT(x) = x  
  
#else  
  
#define GLOBAL extern  
#define INIT(x)  
  
#endif  
  
/* глобальные переменные */  
GLOBAL int variable1 INIT(1);
```

Если DEFINE_GLOBALS определена, то последняя строка вышеприведенного примера выглядит так:

```
int variable1 = 1;
```

Если DEFINE_GLOBALS не определена, то последняя строка приведенного выше примера имеет такой вид:

```
extern int variable1;
```

Не забудьте определить все глобальные переменные Вашего приложения с префиксом GLOBAL, как показано выше. Также убедитесь, что в один (и только в один) из Ваших файлов с исходным кодом включена следующая строка:

```
#define DEFINE_GLOBALS
```

Включение этой строки обеспечивает, что глобальные переменные определены в этом файле и используются как внешние декларации во всех остальных файлах с исходным кодом.

☞ В файле Makefile сборка файлов с исходным кодом зависит от хедер-файла. Так что если Вы внесли какие-либо изменения в хедер-файл, при сборке Вашего приложения будут перекомпилированы все файлы с исходным кодом.

Имена функций и имена файлов

PhAB генерирует функцию для каждой функции инициализации, установочной функции модуля, ответной реакции, функции пункта меню и прочего, что Вы задаёте в Вашем приложении. Если Вам не нужна функция, оставьте пустым её имя.

После того как функция сгенерирована, Вы вольны модифицировать её. Есть только одно условие: если Вы изменили имя функции, Вы должны также изменить имя, которое Вы задали в связанной ответной реакции или определении внутренней связи. В противном случае PhAB будет продолжать регенерировать старое имя каждый раз при генерации приложения.

Способ, которым Вы задаёте имя функции в PhAB, определяет имя файла заготовки:

- `function_name` Создаёт заготовку файла на C под именем `function_name.c`
- `function_name@filename.ext` Создаёт заготовку функции и помещает её в `filename.ext`. Этот файл будет включать хедеры и структуру функции, требуемые для компиляции в окружении Photon'a.

☞ PhAB опознает расширения `.cc`, `.cpp` и `.C` как расширения языка C++.

Если это файл уже существует, заготовка функции добавляется к нему. Вы можете использовать эту технологию для уменьшения количества файлов с исходным кодом в Вашем приложении. Вы можете разместить любое количество функций в одном файле. Мы рекомендуем Вам помещать все функции, относящиеся к модулю, в одном файле.

- `function_name.ext` Короткая форма для `function_name@function_name.ext`
- `class::function_name@filename.cc`

Генерирует заготовку функции статического члена C++, а не прототип.

- `class::function_name@` Не создаёт заготовку функции или прототип. Вместо этого вызывает функцию статического члена класса C++. Для функции члена класса прототипы не генерируются; Ваше приложение должно иметь необходимые декларации в своём глобальном хедер-файле.
- `function_name@` Генерирует прототип C-функции, а не заготовку. Это полезно, если Вы используете библиотеку функций C.
- `::function_name@` Генерирует прототип для функции C++, а не заготовку.

Это полезно, если Вы используете библиотеку функций C++.

Вы можете использовать в одном и том же приложении PhAB'a и C и C++. См. раздел "Что генерирует PhAB" в главе "Генерирование, компилирование и запуск программного кода на исполнение".

Функция инициализации

PhAB позволяет Вам определить функцию инициализации уровня приложения. PhAB API вызывает эту функцию один раз при запуске приложения, перед тем как будут созданы какие-либо окна или другие виджеты. Для получения более полной информации по установкам этой функции см. раздел "Задание информации по запуску приложения" в главе "Работа с приложениями".

Функция инициализации включает стандартные аргументы `argc` и `argv`, так что Ваше приложение может, если необходимо, выполнить синтаксический анализ опций командной строки (см. ниже раздел "Обработка опций командной строки"). Вы также можете использовать эту функцию, чтобы открыть какую-либо базу данных виджетов (см. главу "Доступ к модулям PhAB из программного кода") или другие файлы данных, специфические для данного приложения.

Вот простая функция инициализации, генерируемая PhAB:

```
/*           Ваше описание           */
/*   AppBuilder Photon Code Lib   */
/*           Version 2.01A   */

/* Стандартные хедеры */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
```

```

/* Инструментальные хедеры */
#include <Ph.h>
#include <Pt.h>
#include <Ap.h>

/* Локальные хедеры */
#include "abimport.h"
#include "proto.h"

/* Строка опций приложения */
const char ApOptions[] =
    AB_OPTIONS "";          /* Добавьте Ваши опции в "" */

int
init( int argc, char *argv[] )
{
    /* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    argc = argc, argv = argv;

    /* Обработка опций командной строки - если требуется */
    /* Не забудьте проигнорировать первую обработку Photon'ом */

    /* Типичное место для открытия баз данных виджета */

    /* Любая другая специфическая для приложения инициализация */

    return( Pt_CONTINUE );
}

```

Обработка опций командной строки

Приложение PhAB имеет по умолчанию несколько опций командной строки:

-h height [%] Высота окна в пикселях или процентах от высоты экрана, если заданы %.
-s server_name Имя сервера Photon:

Если server_name является:	Используется этот сервер:
node_number – номер узла	//node_number/dev/photon
fullpath – полный путь	fullpath
relative_path – относительный путь	/dev/relative_path

-w width [%] Ширина окна в пикселях или процентах от ширины экрана, если заданы %.
-x position [%] [r] Координата x верхнего левого угла окна в пикселях или в процентах от ширины экрана, если заданы %. Если задана r, координата является относительной к текущей консоли.
-y position [%] [r] Координата y верхнего левого угла окна в пикселях или в процентах от высоты экрана, если заданы %. Если задана r, координата является относительной к текущей консоли.
-Si|m|n Состояние инициализации главного окна (свёрнутое в иконку, максимизированное или нормальное).

Вы можете запретить опции для размеров и позиции приложения – см. раздел "Опции командной строки" в главе "Работа с приложениями". Вы можете также определить дополнительные опции.

☞ Для включения каких-либо дополнительных опций редактируйте сообщения использования приложения, которые Вы найдёте в файле Usemsg в директории src Вашего приложения. Подробности о синтаксисе сообщений об использовании см. в описании usemsg в "Справочнике утилит QNX 6".

Для обработки опций командной строки используйте функцию getopt(), описанную в "Справочнике функций библиотеки C". В следующем примере показано, как Вы можете обработать несколько опций (три из которых имеют аргументы):

```

const char ApOptions[] = AB_OPTIONS "a:b:c:pqr";

int init( int argc, char *argv[] ) {
    int opt;
    while ( ( opt = getopt( argc, argv, ApOptions ) ) != -1 )
        switch ( opt ) {
            case 'a' : ...
            case 'b' : ...
            ...
            case '?' : ...
        }
    ...
    return Pt_CONTINUE;
}

```

AB_OPTIONS является макросом, который задаёт принимаемые по умолчанию опции, добавляемые PhAB'ом. Он генерируется PhAB'ом на основе Ваших установок запуска приложения. Например, если Вы установили кнопку "No Pos Arg" макрос AB_OPTIONS не будет содержать "x:" или "y:". Вы можете обрабатывать опции в AB_OPTIONS двумя способами:

- включить ветку "case" для каждой опции, но ничего в них не выполнять. Вы можете также включить "default", который будет печатать сообщение при получении неверных опций
или
- не включать ветви "case" для них. Если Вы это сделаете, Вы не можете иметь ветку "default". Библиотека PhAB также просматривает массив ApOptions, чтобы принимать в расчёт добавленные Вами опции. Например, в вышеприведенном коде библиотека опознаёт, что rx100 задаёт позицию по X (вместо с -p), в то время как -ax100 опознано не будет.

Установочные функции модуля

Установочная функция модуля генерируется, если Вы задали имя функции в привязанной ответной реакции модульного типа, как описано в разделе "Ответные реакции модульного типа" в главе "Редактирование ресурсов и ответных реакций в PhAB".

Все установочные функции PhAB имеют три главных аргумента:

```

int base_setup( PtWidget_t *link_instance,
                ApInfo_t *apinfo,
                PtCallbackInfo_t *cbinfo )
{
    /* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    link_instance = link_instance,
    apinfo = apinfo,
    cbinfo = cbinfo;

    /* Вот здесь Ваш код */

    return( Pt_CONTINUE );
}

```

где:

link_instance Указатель на экземпляр созданного модуля PhAB. Вам понадобится сохранить этот указатель, если он указывает на модуль окна, поддерживающего множественность экземпляров.

apinfo Указатель на структуру ApInfo_t, которая обеспечивает:

- указатель на виджет, послуживший причиной вызова установочной функции (т.е. виджет, послуживший причиной того, что отобразился модуль). Для внутренней связи этот указатель является копией указателя на виджет, переданный функции ApCreateModule(); этот указатель полезен при

позиционировании модуля. Вы можете также определить виджет, послуживший причиной вызова установочной функции, вызвав функцию `ApWidget()`.

Обсуждавшиеся коды, связанные с типом вызванной установочной функции:

`ABR_PRE_REALIZE` – эта установочная функция была вызвана перед реализацией модуля

`ABR_POST_REALIZE` эта установочная функция была вызвана после того, как – модуль отобразился на экране.

cbinfo

Указатель на общую структуру ответной реакции `Photon`. Эта структура обеспечивает информацией, связанной с виджетом, вызвавшим ответную реакцию, с событием `Photon'a` и некоторыми данными по ответной реакции, специфическими для виджета. Формат данных варьируется в соответствии с классом виджета и типом ответной реакции. Для получения более полной информации см. описание `PtCallbackInfo_t` в "Справочнике виджетов".

Обычно установочная функция возвращает значение `Pt_CONTINUE`. Это указывает PhAB API продолжать исполнение и отобразить связанный с этой функцией модуль. Для модулей окна и диалога Вы можете возвращать `Pt_END`, чтобы завершить привязанную ответную реакцию и разрушить модуль, не отображая его. Для модулей окна Вы можете возвращать `Pt_HALT`, чтобы не реализовать, но и не разрушать окно. Это является полезным, если Вы хотите реализовать окно позже.

Функции ответных реакций кодового типа

Функция ответной реакции кодового типа генерируется, если Вы задаёте связанную ответную реакцию кодового типа, как описано в разделе "Кодовые ответные реакции" главы "Редактирование ресурсов и ответных реакций в PhAB".

Все ответные реакции кодового типа имеют три главных аргумента:

```
int mycallback( PtWidget_t *widget,
               ApInfo_t *apinfo,
               PtCallbackInfo_t *cbinfo )
{
    /* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    widget = widget,
    apinfo = apinfo,
    cbinfo = cbinfo;

    /* Вот здесь Ваш код */

    return( Pt_CONTINUE );
}
```

где

widget Указатель на экземпляр виджета, вызвавшего ответную реакцию. Это указатель на структуру `PtWidget_t`, но Вы можете интерпретировать его как идентификатор виджета; не пытайтесь манипулировать членами структуры.

apinfo Указатель на структуру `ApInfo_t`, которая включает коды, связанные с типом функции ответной реакции, которая была вызвана:

`ABR_CANCEL` – эта функция ответной реакции вызывается связью "Cancel"

`ABR_CODE` – эта функция ответной реакции вызывается связью "Code"

`ABR_DONE` – эта функция ответной реакции вызывается связью "Done".

cbinfo Указатель на общую структуру ответной реакции `Photon'a`. Структура обеспечивает информацией, связанной с виджетом, вызвавшим ответную реакцию, с событием `Photon'a` и некоторыми данными по ответной реакции, специфическими для виджета. Формат данных варьируется в соответствии с классом виджета и типом ответной реакции. Для получения более полной информации см. описание `PtCallbackInfo_t` в "Справочнике виджетов".

Ваша ответная реакция должна возвращать `Pt_CONTINUE`, если описание ответной реакции не даст Вам повода вернуть что-то другое. Ответные реакции `ABR_CANCEL` и `ABR_DONE` могут возвращать `Pt_HALT`, чтобы не допустить закрытия модуля.

Типы данных геометрии

Вот структура данных, которую Вы будете использовать при задании позиции, размеров или областей:

<code>PhPoint_t</code>	Координаты <i>x</i> и <i>y</i> одиночной точки. Вы будете использовать это при задании расположения на экране, на холсте виджета, и прочем.
<code>PhDim_t</code>	Ширина (<i>w</i>) и высота (<i>h</i>), обычно в координатах Photon'a. Вы будете это использовать при задании размеров.
<code>PhArea_t</code>	Прямоугольная область, выраженная как <code>PhPoint_t</code> для левого верхнего угла области и <code>PhDim_t</code> , определяющего размеры области.
<code>PhRect_t</code>	Прямоугольник, выраженный двумя структурами <code>PhPoint_t</code> , одна для верхнего левого, вторая – для нижнего правого углов.
<code>PhTile_t</code>	Список прямоугольников. Эта структура используется в основном в "списке повреждений", который определяет области на экране или виджет, которые должны быть обновлены.

☞ Photon поддерживает внутренний пул "черепиц", поскольку они часто используются, и использование пула уменьшает время, затрачиваемое на удаление "черепиц" и освобождение ресурсов. Используйте `PhGetTile()`, чтобы получить "черепицу" из пула, и `PhFreeTiles()`, чтобы вернуть список "черепиц" в пул.

Вы, вероятно, не будете использовать структуру `PhTile_t`, если не будете использовать виджет `PtRaw` или создавать свой собственный виджет. Для получения более полной информации см. раздел "Виджет `PtRaw`" в главе "Необработанная прорисовка и анимация", и "Построение своих виджетов".

Библиотеки Photon'a предоставляют различные функции работы с этими типами данных:

<code>PhAreaToRect()</code>	Преобразование области в прямоугольник
<code>PhDeTranslateRect()</code>	Детрансляция прямоугольника (вычитание смещения)
<code>PhRectIntersect()</code>	Нахождение перекрытия двух прямоугольников
<code>PhRectToArea()</code>	Преобразование прямоугольника в область
<code>PhRectUnion()</code>	Определение охватывающего прямоугольника для двух прямоугольников
<code>PhTranslateRect()</code>	Трансляция прямоугольника (добавление смещений)

Таймеры

Если Вы желаете диспетчировать Ваши собственные операции в конкретные интервалы времени, или если Вы желаете выполнять тайм-аут или запускать событие именно в конкретный момент времени, Вам могут понадобиться основанные на таймере функции ответных реакций. Есть несколько способов это осуществить, с различным объёмом сложности и точности:

- виджет `PtTimer` – прост, но не слишком точен;
- функция `RtTimer*` – несколько больше работы, несколько более точная;

- таймеры в процессе, отдельном от GUI (графический интерфейс пользователя, кто забыл) – необходим для жёсткого реального времени. Для получения более полной информации см. раздел "Нити" в главе "Переменные операции".

Библиотеки Photon'a также включают несколько функций работы с таймером низкого уровня, но Вам надо пользоваться ими с осторожностью:

PhTimerArm() Взводит событие таймера. Не используйте эту функцию в приложении, которое использует виджеты.

PtTimerArm() Взводит событие таймера в виджете. Эта функция обычно используется при создании своего собственного виджета. Некоторые виджеты (такие как PtTerminal) уже используют этот тип таймера, так что вызов функции PtTimerArm() может иметь непредсказуемые результаты.

Использование PtTimer

Простейшим способом обеспечить работу таймера является использование виджета PtTimer. Он определяет такие ресурсы:

Pt_ARG_TIMER_INITIAL	– начальный интервал истечения времени
Pt_ARG_TIMER_REPEAT	– необязательный интервал времени повтора
Pt_CB_TIMER_ACTIVATE	– время окончания ответной реакции

Для получения более полной информации см. "Справочник виджетов".

☞ Когда Вы создаёте виджет PtTimer в PhAB, он возникает как чёрный прямоугольник. Этот прямоугольник не появится, когда Вы запустите приложение – это просто заполнитель места [Малевичи... Прим.пер.].

PtTimer прост в использовании, но не даёт точности событий таймера. В частности, он не гарантирует постоянной частоты повторения; поскольку повторение осуществляется путём взведения таймера вновь для каждого события, какие-либо задержки в обработке событий аккумулируются. Таймеры ядра гарантируют точность частоты повторения, даже если Ваше приложение не может её выдерживать.

Функции RtTimer*

Функции RtTimer* (описанные в "Справочнике библиотеки Photon") дают более точный отсчёт времени, нежели PtTimer, но ещё не соответствуют требованиям жёсткого реального времени. Они охватывают функции POSIX, манипулирующие таймерами ядра:

RtTimerCreate()	Создать таймера реального времени
RtTimerDelete()	Удалить таймера реального времени
RtTimerGetTime()	Получить у таймера реального времени оставшееся время
RtTimerSetTime()	Установить время срабатывания для таймера реального времени

Эти функции более точные, чем PtTimer, поскольку таймер взводится по-новому не Photon'ом, а ядром. Однако, если Photon занят обработкой событий, это по-прежнему может привести к задержке в обработке произошедших событий.

Меню инициализации

Вам может понадобиться сделать с меню различные вещи до того, как оно будет отображено. Вы можете использовать установочную функцию меню, чтобы:

- включить, отключить или переключить пункты меню
- изменить текст для какого-то пункта

Вы можете также использовать функциональный пункт меню, чтобы сгенерировать новые пункты во время исполнения.

Методы, выполняющие это, обсуждаются в нижеследующих разделах.

Включение, отключение или переключение пунктов меню

Если в текущем состоянии пункт меню не является допустимым выбором, хорошей идеей является отключить его, так чтобы пользователь и не пытался его выбрать. Конечно, Вам также понадобится сделать его доступным, когда настанет время. Если Ваше меню имеет какие-либо переключающиеся пункты, Вам также понадобится установить их перед тем, как меню будет отображено. Чтобы выполнить это, используйте функцию `ApModifyItemState()`. Функция `ApModifyItemState()` принимает переменное число аргументов:

- Первым аргументом является указатель на модуль меню. Например, если именем экземпляра модуля меню является `draw-menu`, передайте `&draw-menu` как первый параметр.
- Вторым параметром является желаемое состояние:

<code>AB_ITEM_DIM</code>	для отключения пункта
<code>AB_ITEM_NORMAL</code>	для включения и возвращения пункта в исходное состояние
<code>AB_ITEM_SET</code>	для установки переключающегося пункта
- Остальные аргументы формируют завершающийся `NULL`'ом список пунктов меню, которые будут установлены в задаваемое состояние. Этот список состоит из глобальных переменных `ABN_...`, относящихся к пунктам меню.

Например, предположим, что Ваше приложение имеет модуль меню по имени `draw_menu`, включающее пункты с именами экземпляров `draw_group` и `draw_align`. Мы можем отключить эти пункты следующим вызовом:

```
ApModifyItemState( &draw_menu,          AB_ITEM_DIM,
                  ABN_draw_group,      ABN_draw_align,  NULL);
```

Изменение текста пунктов меню

Для изменения текста пункта меню, например, замены команды на её противоположную, Вы можете использовать функцию `ApModifyItemText()`. Её аргументы следующие:

- указатель на модуль меню. Например, если имя экземпляра объекта модуля меню – `draw_menu`, передайте в качестве первого параметра `&draw_menu`.
- глобальная переменная `ABN_...` для пункта меню
- новый текст

Например, наше меню "Draw" может иметь пункт, который должен быть либо "Group" (сгруппировать), либо "Split" (разгруппировать), в зависимости от того, какой объект выбрал пользователь. Мы можем изменить текст пункта `draw_group` в меню `draw_menu` с помощью следующего кода:

```
ApModifyItemText(&draw_menu, ABN_draw_group, "Split");
```

Чтобы получить текущий текст пункта, вызовите функцию `ApGetItemText()`.

Генерирование пунктов меню

Иногда Вам может потребоваться генерировать пункты меню в ходе выполнения задачи. Например, меню "Window" PhAB'a включает список модулей в Вашем приложении. Чтобы сгенерировать пункты меню, добавьте пункт-функцию в Ваш модуль меню (как описано в разделе "Создание пунктов-функций" в главе "Работа с модулями") и отредактируйте сгенерированную PhAB'ом заготовку функции.

Например, если Ваш модуль `draw_menu` включает пункт-функцию, который вызывает функцию `add_shapes()`, PhAB сгенерирует следующий код:

```
int add_shapes (PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo)
{
    /* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    widget=widget, apinfo=apinfo, cbinfo=cbinfo;

    return (Pt_CONTINUE);
}
```

Параметры, переданные этой функции, не используются. Мы используем функцию `PtCreateWidget()`, чтобы создать пункты меню, которые обычно являются виджетами `PtMenuButton`. Как описано в главе "Манипулирование ресурсами в программном коде приложения", чтобы установить начальные значения для ресурсов, мы можем использовать тот же тип списка аргументов, который мы использовали с функцией `PtSetResources()`. Например, чтобы добавить пункт "Rectangle" с клавишей быстрого доступа "R":

```
PtArg_t    args[2];
PtWidget_t *new_item;

PtSetArg (&args[0], Pt_ARG_TEXT_STRING, "Rectangle", 0);
PtSetArg (&args[1], Pt_ARG_ACCEL_KEY, "R", 0);
new_item = PtCreateWidget( PtMenuButton, Pt_DEFAULT_PARENT, 2, args);
```

Вторым параметром в вызове `PtCreateWidget()` является родитель виджета; когда Вы генерируете пункты меню, он должен быть установлен в `Pt_DEFAULT_PARENT`. Это делает новый пункт потомком текущего меню или подменю. Не вызывайте в этом случае функцию `PtSetParentWidget()`. Далее, мы хотим прикрепить к новому пункту функцию ответной реакции. Мы должны сделать это вручную; PhAB не создаёт для этого заготовку функции. Например, ответная реакция для нашего нового пункта может быть такой:

```
int rect_callback ( PtWidget_t *widget, void *client_data, PtCallbackInfo_t *cbinfo)
{
    .....
}
```

Эта ответная реакция похожа на код ответной реакции, генерируемой PhAB. Её аргументами являются:

widget Указатель на выбранный пункт меню

client_data Произвольные данные, передаваемые ответной реакцией.

☞ Это отличается от кода ответной реакции PhAB, которой в качестве второго аргумента передаётся `apinfo`.

cbinfo Указатель на общую структуру ответной реакции Photon'a. Структура предоставляет информацию, относящуюся к виджету, вызвавшему ответную реакцию, событие Photon'a и некоторые данные по ответной реакции, специфические для виджета. Формат данных зависит от класса виджета и типа ответной реакции. Для получения более полной информации см. описание `PtCallbackInfo_t` в "Справочнике виджетов".

Последнее, что нам надо сделать – это добавить ответную реакцию в список ответных реакций пункта меню `Pt_CB_ACTIVATE`, используя функцию `PtAddCallback()`:

```
PtAddCallback(new_item, Pt_CB_ACTIVATE, rect_callback, NULL);
```

Последний аргумент в вызове функции `PtAddCallback()` задаёт то, что будет передано как аргумент `client_data` ответной реакции. Для получения более полной информации см. раздел "Ответные реакции" в главе "Управление виджетами в программном коде приложения".

Давайте сведём всё это вместе:

```
int rect_callback( PtWidget_t *widget, void *client_data, PtCallbackInfo_t *cbinfo)
{
    ...
}

int
add_shapes (PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo)
{
    PtArg_t    args[2];
    PtWidget_t *new_item;

    /* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    widget=widget, apinfo=apinfo, cbinfo=cbinfo;

    PtSetArg (&args[0], Pt_ARG_TEXT_STRING, "Rectangle", 0);
    PtSetArg (&args[1], Pt_ARG_ACCEL_KEY, "R", 0);
    new_item = PtCreateWidget( PtMenuButton, Pt_DEFAULT_PARENT, 2, args);
    PtAddCallback (new_item, Pt_CB_ACTIVATE, rect_callback, NULL);

    /* Повторить всё вышеприведенное для других форм */

    return (Pt_CONTINUE);
}
```

Создание подменю

Вы можете создать подменю в меню, созданном для функции-пункта меню, как показано ниже:

1. Создать кнопку меню для каскадного меню, установив Pt_ARG_BUTTON_TYPE в Pt_MENU_RIGHT или Pt_MENU_DOWN, как потребуется.
2. Сохранить указатель на текущий родительский виджет, вызвав функцию Pt_GetParent():

```
menu=PtGetParentWidget();
```

3. Создать новый виджет PtMenu и установить Pt_MENU_CHILD в ресурсе нового меню Pt_ARG_MENU_FLAGS.
☞ PtMenu является контейнером, так что это новое меню становится текущим принятым по умолчанию родителем.
4. Создать пункты подменю, как описано выше.
5. Восстановить принимаемого по умолчанию родителя из сохранённого значения, вызвав PtSetParentWidget():

```
PtSetParentWidget(menu);
```

6. Продолжить добавление пунктов в верхнем меню, если требуется.

Этот пример показывает, как генерировать подменю, а также как client_data могут быть использованы для ответных реакций класса для идентификации пункта, выбранного из меню:

```
/* Меню с подменю */

/* Стандартные хеадеры */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

/* Инструментальные хеадеры */
#include <Ph.h>
#include <Pt.h>
#include <Ap.h>

/* Локальные хеадеры */
#include "abimport.h"
#include "proto.h"

/* Константы для форм в меню */
#define RECTANGLE 1
#define CIRCLE 2
```

```

#define DOT      3
#define BLOB    4
#define POLYGON 5

int ShapeMenuCB( PtWidget_t *widget, void *client_data, PtCallbackInfo_t *cbinfo )
{
    int shape_chosen = (int) client_data;

    widget=widget, client_data=client_data, cbinfo=cbinfo;

    /* Эта ответная реакция использует клиентские данные, чтобы */
    /* определить, какая форма была выбрана */

    switch (shape_chosen) {

        case RECTANGLE: ...
            break;
        case CIRCLE : ...
            break;
        case DOT : ...
            break;
        case BLOB : ...
            break;
        case POLYGON : ...
            break;
        default : printf ("Неизвестная форма: %d\n", shape_chosen);
    }

    return (Pt_CONTINUE);
}

int add_shapes( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo )
{
    PtArg_t args[3];
    PtWidget_t *menu, *new_item;

    /* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    widget = widget, apinfo = apinfo, cbinfo = cbinfo;

    PtSetArg (&args[0], Pt_ARG_TEXT_STRING, "Rectangle", 0);
    PtSetArg (&args[1], Pt_ARG_ACCEL_KEY, "R", 0);
    new_item = PtCreateWidget( PtMenuButton, Pt_DEFAULT_PARENT, 2, args);
    PtAddCallback ( new_item, Pt_CB_ACTIVATE, ShapeMenuCB, (void *) RECTANGLE );

    PtSetArg (&args[0], Pt_ARG_TEXT_STRING, "Circle", 0);
    PtSetArg (&args[1], Pt_ARG_ACCEL_KEY, "C", 0);
    new_item = PtCreateWidget( PtMenuButton, Pt_DEFAULT_PARENT, 2, args);
    PtAddCallback ( new_item, Pt_CB_ACTIVATE, ShapeMenuCB, (void *) CIRCLE );

    /* Создание кнопки меню для подменю */

    PtSetArg (&args[0], Pt_ARG_TEXT_STRING, "Miscellaneous", 0);
    PtSetArg (&args[1], Pt_ARG_ACCEL_KEY, "M", 0);
    PtSetArg (&args[2], Pt_ARG_BUTTON_TYPE, Pt_MENU_RIGHT, 0 );
    new_item = PtCreateWidget( PtMenuButton, Pt_DEFAULT_PARENT, 3, args);

    /* Сохранение текущего принятого по умолчанию родителя */

    menu = PtGetParentWidget();

    /* Создание подменю. Оно становится новым принимаемым по умолчанию родителем */

    PtSetArg (&args[0], Pt_ARG_MENU_FLAGS, Pt_MENU_CHILD, Pt_MENU_CHILD);
    new_item = PtCreateWidget( PtMenu, Pt_DEFAULT_PARENT, 1, args);

    /* Добавление пунктов в подменю */

    PtSetArg (&args[0], Pt_ARG_TEXT_STRING, "Dot", 0);
    PtSetArg (&args[1], Pt_ARG_ACCEL_KEY, "D", 0);
    new_item = PtCreateWidget( PtMenuButton, Pt_DEFAULT_PARENT, 2, args);
    PtAddCallback ( new_item, Pt_CB_ACTIVATE, ShapeMenuCB, (void *) DOT );

    PtSetArg (&args[0], Pt_ARG_TEXT_STRING, "Blob", 0);
    PtSetArg (&args[1], Pt_ARG_ACCEL_KEY, "B", 0);
    new_item = PtCreateWidget( PtMenuButton, Pt_DEFAULT_PARENT, 2, args);
    PtAddCallback ( new_item, Pt_CB_ACTIVATE, ShapeMenuCB, (void *) BLOB);

    /* Восстановление текущего принятого по умолчанию родителя */

    PtSetParentWidget (menu);
}

```

```
/* Продолжение добавления пунктов в верхнее меню */  
  
PtSetArg (&args[0], Pt_ARG_TEXT_STRING, "Polygon", 0);  
PtSetArg (&args[1], Pt_ARG_ACCEL_KEY, "P", 0);  
new_item = PtCreateWidget( PtMenuButton, Pt_DEFAULT_PARENT, 2, args);  
PtAddCallback ( new_item, Pt_CB_ACTIVATE, ShapeMenuCB, (void *) POLYGON);  
  
return( Pt_CONTINUE );  
}
```

Задержка и принудительное обновление изображения на экране

Если Ваше приложение выполняет одновременно изменения в большом количестве виджетов, Вы можете захотеть задержать обновление экрана до тех пор, пока не завершатся все изменения. Это может дать уменьшение мерцания, и в некоторых случаях улучшить производительность Вашего приложения.

Вы можете задержать обновление:

- всех виджетов Вашего приложения – библиотеки Photon'a записывают какие-либо изменения, но не перерисовывают виджеты.
- заданного контейнера и его потомков – библиотеки даже не записывают изменения.

Глобально

Библиотеки Photon'a используют "счётчик удержаний", позволяющий Вам задерживать обновление экрана для Вашего приложения в целом:

- Когда счётчик удержаний ненулевой, экран не обновляется. Для увеличения значения счётчика удержаний вызывается функция `PtHold()`.
- Когда Вы модифицируете виджет, библиотеки помечают его как "повреждённые".
- Когда счётчик удержаний равен 0, библиотеки восстанавливают повреждённые виджеты как нормальные. Чтобы уменьшить значение счётчика удержаний, вызываются функции `PtRelease()` или `PtUpdate()` (эти две функции идентичны).

Для получения более полной информации об этих функциях см. "Справочник библиотечных функций Photon'a".

Для конкретного контейнера

Библиотеки Photon'a используют "счётчик изменений", позволяющий Вам задерживать обновление изображения для конкретного контейнера. Когда счётчик изменений ненулевой и Вы модифицируете контейнер или его потомков, виджеты не помечаются как "повреждённые". Что происходит, когда значение счётчика изменений опять возвращается в ноль, зависит от того, какие функции Вы используете:

```
PtStartFlux()  
PtEndFlux()
```

Когда счётчик изменений контейнера приходит в ноль, Вы должны явно повредить области, которые хотите восстановить.

```
PtContainerHold()  
PtContainerRelease()
```

Когда счётчик изменений контейнера приходит в ноль, весь контейнер в целом помечается как повреждённый.

PtContainerHold() и PtContainerRelease() более простые в использовании, поскольку Вам не надо определять, какие виджеты или области Вам необходимо повреждать. Однако это может приводить к большему мерцанию, чем в случае использования функций PtStartFlux() и PtEndFlux(). Если Вам необходимо определить, находится ли контейнер или любой из его родителей в текущий момент в изменении, используйте PtIsFluxing().

Для получения более полной информации по этим функциям см. "Справочник библиотечных функций Photon'a".

Принудительное обновление

Для немедленного обновления повреждённых областей изображения Вы можете в любой момент использовать вызов функции PtFlush(). Функция PtFlush() игнорирует счётчик удержаний и не изменяет его значение. Если контейнер находится в изменении и Вы модифицируете его или его потомков, библиотеки Photon'a не помечают виджеты как повреждённые, так что функция PtFlush() не будет их восстанавливать.

- ☞ Комбинирование удержаний для всего приложения, удержание контейнеров и вызовов функции PtFlush() может дать непредсказуемый результат. Например, если Вы удерживаете всё приложение, повреждаете часть контейнера, удерживаете контейнер, модифицируете его, и затем вызываете функцию PtFlush(), библиотеки восстановят повреждение – отобразив какую бы то ни было порцию модификации, которая оказала воздействие на повреждённую область.

Глава 10. Манипулирование ресурсами в коде приложения

В этой главе описано, как Вы можете устанавливать и получать значения ресурсов виджетов из Вашего приложения. Глава включает:

- Списки аргументов
- Установка ресурсов
- Получение ресурсов

Хотя Вы и можете устанавливать начальные значения ресурсов виджетов в PhAB'e, Вам, вероятно, понадобится получить доступ к ним из своего кода. Например:

- когда появляется диалог, Вам может понадобиться предварительно инициализировать какие-то отображаемые данные, так что Вы будете устанавливать ресурсы;
- когда пользователь введёт значение в виджете PtText, набрав некий текст, Вам, возможно, понадобится это значение в Вашей программе, так что Вы будете получать ресурсы.

Дополнительно, если Вы используете функцию PtCreateWidget(), чтобы приписать значение (instantiate) виджету в Вашем коде, Вы можете задавать его ресурсам какие-то первоначальные значения.

Значения ресурса устанавливаются и извлекаются путём использования списка аргументов. Вот два шага, приводящие к установке или извлечению более чем одного значения ресурса:

- ☞ • Установка списка аргументов с использованием макроса PtSetArg()
- Установка значения с использованием функции PtSetResources(), или извлечение значения с помощью функции PtGetResources().

Если Вы извлекаете или устанавливаете один ресурс, проще использовать функции PtGetResource() или PtSetResource() – в этом случае Вам нет необходимости устанавливать список аргументов.

Список аргументов

Список аргументов – это массив структур PtArg_t (см. "Справочник библиотечных функций Photon"). Каждый из этих элементов определяет ресурс виджета и новое значение для ресурса (или адрес значения, которое будет устанавливать текущее значение ресурса).

Чтобы инициализировать каждый элемент списка аргументов, можно использовать макрос PtSetArg():

```
PtSetArg( PtArg_t .arg, long type, long value, long len);
```

- ☞ • Если значение не требуется вычислять во время исполнения, Вы можете для инициализации списка аргументов использовать вместо этого функцию Pt_ARG(). Для получения более полной информации см. "Справочник библиотечных функций Photon".

Первые два аргумента в PtSetArg() являются адресом элемента списка аргументов и именем ресурса. Третий и четвёртый аргументы варьируются в зависимости от типа ресурса и используются либо для установки, либо для получения значения ресурса. Когда ресурс устанавливается, третий аргумент всегда используется для хранения значения ресурса или адреса на значение ресурса. Четвёртый аргумент используется либо как указатель размера, либо как

маска, в зависимости от типа определяемого значения. Ниже в таблице приведены возможные типы ресурсов:

ТИП	ОПИСАНИЕ
Alloc	Объект памяти произвольного размера
Array	Массив
Boolean	Бит, который может быть либо выставлен, либо сброшен
Color	Цвет
Complex	Ресурс, который обрабатывается особым образом; см. ниже
Flag	Значение, в котором каждый бит имеет свой смысл
Function	Указатель на функцию
Image	Указатель на структуру PhImage_t
Link	Связанный список
Pointer	Указатель на адрес, который Вы определяете
Scalar	Значение, которое может быть представлено в одном long
String	Завершающаяся NULL-ом строка
Struct	Тип данных фиксированного размера, обычно структура, float или double

Для получения информации по ресурсам, оговоренным для каждого виджета, см. "Справочник виджетов Photon'a".

☞ Ресурсы complex являются особыми; для получения инструкций по их установке и получению значения см. их описание в "Справочнике виджетов Photon'a". Виджеты, имеющие комплексные ресурсы, обычно имеют функции, облегчающие работу с ними.

Установка ресурсов

Помните, что есть два шага, связанные с установкой значений более чем одного ресурса:

- Установка списка аргументов с использованием макроса PtSetArg()
- Установка значения с использованием функции PtSetResources().

Если Вы устанавливаете один ресурс, проще использовать PtSetResource() – Вам нет нужды устанавливать список аргументов. См. раздел "Установка одного ресурса" ниже.

Списки аргументов для установки ресурсов

Многие нижеследующие разделы демонстрируют установку неких ресурсов для виджета PtComboBox. Заметьте, что Вы можете установить одновременно более одного ресурса. Чтобы сделать это, определите список аргументов соответствующей длины:

```
PtArg_t args[5];
```

После инициализации списка аргументов, Вы фактически установили ресурсы.

Ресурсы scalar и color

При установке скалярного ресурса Вы задаёте значение как третий аргумент в PtSetArg(). Четвёртый аргумент не используется и должен быть установлен в 0. Например, чтобы установить ширину фаски в виджете типа "combo box", передайте новое значение как третий аргумент:

```
PtSetArg(&args[0], Pt_ARG_BEVEL_WIDTH, 5, 0);
```

Когда Вы вызываете PtSetResources(), виджет копирует скалярное значение в свою собственную внутреннюю структуру данных.

Ресурсы string

Установка строкового значения схожа с установкой скалярного; Вы задаёте строку как третий аргумент макроса PtSetArg(). Четвёртый аргумент – это число копируемых байт; если он равен 0, для определения длины строки используется strlen().

Например, чтобы установить текст по умолчанию для виджета типа "combo box", Вы должны задать значение ресурса Pt_ARG_TEXT_STRING в элементе под номером один списка аргументов:

```
PtSetArg(&args[1], Pt_ARG_TEXT_STRING, "Rectangle", 0);
```

Когда Вы вызываете PtSetResources(), виджет копирует строку в свою собственную внутреннюю структуру данных. Если Вам необходимо использовать международные (не-ASCII) символы в строке [читай – русский текст. Прим. пер.], следуйте одним из этих решений:

- Определите строку в базе данных виджетов и используйте редактор языка для перевода строки. См. главу "Поддержка международных языков".
- Используйте red или иной UTF-совместимый текстовый редактор, чтобы отредактировать код C приложения. Вы можете затем использовать формирующие последовательности, описанные в разделе "Формирующие последовательности Photon'a" приложения "Поддержка многоязычности Unicode".
- ☞ Большинство консольных редакторов, таких как elvis и vedit, не являются UTF-совместимыми. Для получения более подробной информации по редактору red см. книгу "Справочник утилит QNX6".
- Разыщите требуемый символ в <photon/PkKeyDef.h>, используйте wctomb(), чтобы конвертировать символ из Unicode в UTF-8, и затем впишите шестнадцатичный код в Вашу строку. Например, французское слово "rèsumè" будет закодировано как "r\xC3\xa9sum\xC3\xa9" – тяжело для чтения, но зато работает со всеми редакторами. Для получения более полной информации по Unicode и UTF-8 см. приложение "Поддержка многоязычности Unicode".

Ресурсы alloc

Некоторые ресурсы спроектированы для хранения выделенного блока памяти. Например, каждый виджет включает ресурс Pt_ARG_USER_DATA, который Вы можете использовать для хранения каких-то данных, которые Вы хотите иметь во внутренней памяти виджета. Для установки этого ресурса передайте указатель на данные как третий аргумент в PtSetArg(). Четвёртым аргументом является размер блока памяти в байтах:

```
my_struct user_data;
/* Инициализация данных */
PtSetArg(&args[2], Pt_ARG_USER_DATA, &user_data, sizeof(user_data));
```

Когда Вы вызовете PtSetResources(), виджет скопирует данное количество байт в свою внутреннюю память.

Ресурсы image

Ресурсы образов спроектированы для хранения структуры PhImage_t. Например, виджет PtLabel имеет ресурс Pt_ARG_LABEL_IMAGE, который Вы можете использовать для хранения образа. Чтобы установить этот ресурс, создайте и инициализируйте структуру PhImage_t, затем передайте указатель на неё как третий аргумент в PtSetArg(). Четвёртый аргумент должен быть 0:

```
PhImage_t my_image;
/* Создание и инициализация образа */
PtSetArg(&args[2], Pt_ARG_LABEL_IMAGE, my_image, 0);
```

Когда Вы вызовете PtSetResources(), виджет скопирует структуру образа (но не какую-либо память, на которую указывают члены структуры PhImage_t) в свою внутреннюю память.

Ресурсы array

При установке значения массива третьим аргументом в `PtSetArg()` является адрес массива. Четвёртым аргументом является число элементов массива. Например, для установки `Pt_ARG_ITEMS` списка выборов для виджета типа "combo box", может использоваться следующий вход в список аргументов:

```
char cbox_items[3] = {"Circle", "Rectangle", "Polygon"};
PtSetArg(&args[3], Pt_ARG_ITEMS, cbox_items, 3);
```

Когда Вы вызовете `PtSetResources()`, виджет скопирует содержание массива в свою собственную структуру данных.

Ресурсы flag

При установке флага третьим аргументом в `PtSetArg()` является битовая область, задающая значение устанавливаемых битов. Четвёртым аргументом является битовая маска, указывающая, какие элементы битовой области должны использоваться.

☞ В качестве значения используйте `Pt_TRUE`, `Pt_FALSE` или комбинацию определённых битов и их дополнений. Не используйте значения 1, поскольку это содержит просто один бит как таковой; этот бит может не соответствовать биту, который Вы пытаетесь установить.

Например, следующий список аргументов задаёт выставление флага `Pt_COMBOBOX_STATIC` в виджете типа `combo box` (так что `combo box` всегда отображает список пунктов):

```
PtSetArg(&args[4], Pt_ARG_CBOX_FLAGS, Pt_TRUE, Pt_COMBOBOX_STATIC);
```

Когда Вы вызовете `PtSetResources()`, виджет использует битовую маску для определения, какие биты в его внутреннем ресурсе флагов представлены для изменений. Он берёт значения битов из заданного значения.

Ресурсы function

При установке ресурса функции третьим аргументом для функции `PtSetArg()` передаётся указатель функции. Четвёртый аргумент игнорируется; установите его в 0. Например, чтобы задать функцию прорисовки виджета `PtRaw`, установите ресурс `Pt_ARG_RAW_DRAW_F` следующим образом:

```
PtSetArg(&args[0], Pt_ARG_RAW_DRAW_F, &my_raw_draw_fn, 0);
```

Когда Вы вызовете `PtSetResources()`, виджет скопирует указатель на ресурс.

Ресурсы pointer

При установке ресурса указатель должен задаваться третьим аргументом функции `PtSetArg()`. Четвёртый аргумент игнорируется и должен быть установлен в 0.

При вызове `PtSetResources()` виджет просто тупо делает копию указателя в ресурсе.

☞ Виджет не делает копию памяти, на которую ссылается указатель; не освобождайте память, пока виджет ещё ссылается на неё.

Например, каждый виджет включает ресурс `Pt_ARG_POINTER`, который Вы можете использовать для хранения во внутренней памяти виджета указателя на какие-либо произвольные данные. Виджет никогда не обращается к этим данным; это просто для использования им. Чтобы установить этот ресурс, выделите требуемую память и передайте указатель на неё как третий аргумент в `PtSetArg()`. Четвёртый аргумент установите в 0:

```
my_struct user_data;
/* Выделение памяти и инициализация данных */
PtSetArg(&args[0], Pt_ARG_POINTER, user_data, 0);
```

Когда Вы вызываете `PtSetResources()`, виджет копирует значение указателя в свою внутреннюю память.

Ресурсы Link

При установке связанного списка передайте адрес массива данных как третий аргумент в функцию `PtSetArg()`. Четвёртый аргумент имеет определённый специфический смысл:

`num` добавляет `num` пунктов (если `num` равен 0, добавляется один пункт)

`Pt_LINK_INSERT`

вставляет первый элемент массива в начало связанного списка

`Pt_LINK_DELETE`

удаляет первый элемент списка, который совпадает с первым элементом массива

Когда Вы вызываете `PtSetResources()`, виджет копирует данные в свою внутреннюю память.

Ресурсы struct

При установке ресурса структуры передайте функции `PtSetArg()` в качестве третьего аргумента адрес данных. Четвёртый аргумент не используется и должен быть установлен в 0. Когда Вы вызываете `PtSetResources()`, виджет скопирует данные в свою внутреннюю память.

Ресурсы boolean

При установке булевого ресурса Вы должны задать значение как третий аргумент функции `PtSetArg()`, используя 0 как "ложь" и ненулевое значение в качестве "истина". Четвёртый аргумент не используется и должен быть установлен в 0. Например, чтобы задать для `PtTerminal` протокол ANSI, передайте в качестве третьего аргумента ненулевое значение:

```
PtSetArg(&args[1], Pt_ARG_TERM_ANSI_PROTOCOL, 1, 0);
```

При вызове `PtSetResources()` виджет сбрасывает или устанавливает один бит в своей собственной структуре данных в зависимости от того, нулевое или ненулевое значение.

Вызов PtSetResources()

Как только Вы установили список аргументов, всё готово к тому, чтобы устанавливать ресурсы. Помните, что `PtSetArg()` не устанавливает ресурсы; эта функция только устанавливает список аргументов.

Вы можете использовать `PtSetResources()`, чтобы устанавливать новые значения ресурсов:

```
int PtSetResources(PtWidget_t *widget, int n_args, PtArg_t *args);
```

Аргументами этой функции являются указатель на виджет, число входов в список аргументов и сам список аргументов.

Вы можете также установить ресурсы передачей списка аргументов в функцию `PtCreateWidget()`. Правила задания значений в элементах списка аргументов те же самые. Для получения более полной информации см. раздел "Создание виджетов" в главе "Управление виджетами в коде приложения".

Например, Вы можете установить ресурсы виджета типа "combo box", используя список аргументов, созданный выше. Вызовите функцию `PtSetResources()` следующим образом:

```
PtSetResources(ABW_shapes_cbox, 5, args);
```

В ответ на изменение своего ресурса виджет может перерисовать себя. Вызов `PtSetResources()` послужит толчком к этому изменению. Любые изменения, появившиеся в виджете, не будут, однако, иметь эффекта до тех пор, пока не произойдёт восстановление управления в петлю

обработки событий. Таким образом, если функция `PtSetResources()` была вызвана из функции ответной реакции или из функции обработки события, изменения в виджете не будут видны до тех пор, пока не будут выполнены все ответные реакции из списка ответных реакций и все обработчики событий.

Установка одного ресурса

Если Вы устанавливаете один ресурс, проще использовать вместо `PtSetResources()` функцию `PtSetResource()`. Для функции `PtSetResource()` Вам нет нужды устанавливать список аргументов.

Аргументы функции `PtSetResource()` являются комбинацией аргументов `PtSetArg()` и `PtSetResources()`:

```
int PtSetResource(PtWidget_t *widget, long type, long value, long len);
```

widget – это указатель на виджет, ресурс которого мы устанавливаем. Другие аргументы устанавливаются просто так же, как для функции `PtSetArg()`, когда устанавливается более чем один ресурс. См. раздел "Списки аргументов для установки ресурсов" выше.

Например, установка одного ресурса через функцию `PtSetResources()` требует примерно такого кода:

```
PtArg_t args[1];  
PtSetArg(&args[0], Pt_ARG_BEVEL_WIDTH, 5, 0);  
PtSetResources (ABW_shapes_cbox, 1, args);
```

Установка того же ресурса функцией `PtSetResource()` выглядит так:

```
PtSetResource (ABW_shapes_cbox, Pt_ARG_BEVEL_WIDTH, 5, 0);
```

Это выполняется всего одним вызовом функции и не требует никакого массива `args`.

Получение ресурсов

С получением более чем одного значения ресурсов связано два шага:

- Установка списка аргументов с использованием макроса `PtSetArg()`
- Получение значения с использованием функции `PtGetResources()`.

Если Вы получаете один ресурс, проще использовать функцию `PtGetResource()` – Вам не надо устанавливать список аргументов. См. раздел "Получение одного ресурса" ниже. Имеется два метода получения ресурсов: один, связанный с указателями, и второй, с ними не связанный. "Безуказательный" метод обычно проще и безопаснее:

- Так как Вы получаете копию значения, шансы нечаянно переписать настоящее значение уменьшаются.
- Вам не надо заботиться о типе значения (`short` или `long`)
- У Вас имеется небольшое количество локальных переменных и Вы не используете указатели на них, что делает Ваш исходный код легче для восприятия и помогает компилятору генерировать лучший код.

Метод с указателями может быть менее запутанным., если Вы получаете значения одновременно нескольких ресурсов; Вы будете иметь именованные указатели на значения, вместо того, чтобы помнить, какой элемент из списка аргументов какому соответствует ресурсу.

Не используя указатели

Если Вы установили в ноль значения аргументов `value` и `len` при вызове `PtSetArg()`, функция `PtGetResources()` возвращает значение ресурса (преобразованное в `long`) следующим образом:

ТИП РЕСУРСА	VALUE	LEN
Flags (любого типа C)	Значение ресурса	не применяется
Scalar (любого типа C)	Значение ресурса	не применяется
Pointer (любого типа C)	Значение ресурса	не применяется
String	Адрес строки	не применяется
Struct	Адрес данных	не применяется
Array	Адрес первого элемента массива	число членов массива
Alloc	Адрес, где хранится ресурс	не применяется
Boolean	0 (ложь) или 1 (истина)	не применяется

Ресурсы scalar и flags (безуказательный метод)

Чтобы получить скалярный или флаговый ресурс (любого допустимого в C типа) безуказательным методом, выполните следующее:

```
unsigned long getscalar( PtWidget_t *widget, long type ) {
    /* Получение любого вида скаляра */
    PtArg_t arg;
    PtSetArg( &arg, type, 0, 0 );
    PtGetResources( widget, 1, &arg );
    return arg.value;
}
```

Ресурс string (безуказательный метод)

Вот как используется безуказательный метод для получения строчного ресурса:

```
const char *getstr2( PtWidget_t *widget, long type ) {
    PtArg_t arg;

    PtSetArg( &arg, type, 0, 0 );
    PtGetResources( widget, 1, &arg );
    return (char*) arg.value;
}
```

Ресурс boolean (безуказательный метод)

При безуказательном методе получения булевского значения это значение (0 или 1) возвращается в аргументе `value` функции `PtSetArg()`:

```
int getbool( PtWidget_t *widget, long type ) {
    PtArg_t arg;

    PtSetArg( &arg, type, 0, 0 );
    PtGetResources( widget, 1, &arg );
    return arg.value;
}
```

Использование указателей

Когда для получения скалярного ресурса, массива или ресурса флага используется указательный метод, виджет всегда помещает указатель во внутренней структуре данных виджета. В элементе списка аргументов, который Вы устанавливаете использованием функции `PtSetArg()`, Вы должны предоставить адрес переменной, задаваемый указателем внутренних данных.

Четвёртый аргумент в большинстве типов ресурсов не используется. Для массивов это является адресом указателя, который возвращается из `PtGetResources()`, указывая на число входов.

Например, чтобы получить содержимое ресурса `Pt_ARG_FLAGS` (которое есть `long`) для виджета, Вы должны передать адрес указателя как `long`:

```
const long *flags;
PtArg_t arg[1];

PtSetArg(&arg[0], Pt_ARG_FLAGS, &flags, 0);
PtGetResources(ABW_label, 1, arg);
```



`PtGetResources()` возвращает указатели напрямую во внутреннюю память виджета. Не пытайтесь модифицировать ресурсы, непосредственно используя эти указатели. Такая модификация не будет иметь ожидаемого эффекта и вероятно приведёт к ошибкам поведения виджета. Также никогда не освобождайте эти указатели – результатом этого наверняка станет ошибка нарушения памяти или иная подобная ошибка.

Использование указателей `const` поможет избежать этих проблем.

Изменение состояния виджета может сделать эти указатели недействительными; используйте их сразу же.

Если Вы хотите получить значение данного ресурса и затем модифицировать это значение:

1. Получите ресурс
2. Скопируйте ресурс во временную переменную
3. Модифицируйте временную переменную
4. Используя модифицированную копию, установите ресурс

Вы можете использовать полученное значение, чтобы установить значение другого ресурса этого или любого другого виджета, пока Вы не изменили оригинального значения. Например, Вы можете использовать следующий код, чтобы получить `Pt_ARG_TEXT_STRING`, текстовую строку, отображаемую на виджете типа `label` с именем `label`:

```
char *str;
PtArg_t args[1];

PtSetArg(&args[0], Pt_ARG_TEXT_STRING, &str, 0);
PtGetResources(ABW_label, 1, args);
```

Вы можете затем установить эту текстовую строку для другого виджета типа `label` по имени `label2`:

```
PtSetArg(&args[0], Pt_ARG_TEXT_STRING, str, 0);
PtSetResources(ABW_label2, 1, args);
```

Ресурсы `scalar` и `flag` (указательный метод)

Если Вы получаете скалярный ресурс или ресурс флага, используя метод указателя:

- Аргумент `value` в `PtSetArg()` является адресом указателя на соответствующий тип языка C.
- `len` не используется.

Когда вызывается функция `PtGetResources()`, указатель для этого ресурса устанавливается указывающим на внутреннее хранилище виджета.

Вот некоторые функции, которые получают ресурс `scalar` или `flag`, используя метод указателя:

```
unsigned long getlong( PtWidget_t *widget, long type ) {
    /* Получение long или long флагов */
    PtArg_t arg; unsigned long const *result;

    PtSetArg( &arg, type, &result, 0 );
    PtGetResources( widget, 1, &arg );
    return *result;
}

unsigned getshort( PtWidget_t *widget, long type ) {
    /* Получение short или short флагов */
    PtArg_t arg; unsigned short const *result;
```

```
PtSetArg( &arg, type, &result, 0 );
PtGetResources( widget, 1, &arg );
return *result;
}

unsigned getbyte( PtWidget_t *widget, long type ) {
    /* Получение char или char флагов */
    PtArg_t arg; unsigned char const *result;

    PtSetArg( &arg, type, &result, 0 );
    PtGetResources( widget, 1, &arg );
    return *result;
}
```

Ресурсы string (указательный метод)

Если Вы получаете строковые ресурсы, используя метод указателя:

- Аргумент value в PtSetArg() является адресом указателя на char
- len не используется.

Когда вызывается PtGetResources(), задаваемый указатель для строкового ресурса устанавливается указывающим на внутреннее хранилище виджета. Например:

```
const char *getstr1( PtWidget_t *widget, long type ) {
    PtArg_t arg; const char *str;

    PtSetArg( &arg, type, &str, 0 );
    PtGetResources( widget, 1, &arg );
    return str;
}
```

Ресурсы alloc (указательный метод)

Если Вы получаете ресурс размещения, используя метод указателя:

- Аргумент value в PtSetArg() является адресом соответствующего типа (тип определяется по данным, отдаваемым виджету при его установке)
- len не используется.

Когда вызывается PtGetResources(), задаваемый указатель устанавливается указывающим на внутренние данные виджета.

Ресурсы image (указательный метод)

Если Вы получаете ресурсы образа, используя метод указателя:

- Аргумент value в PtSetArg() является адресом указателя на структуру типа PhImage_t
- len не используется.

Когда вызывается PtGetResources(), задаваемый указатель устанавливается указывающим на внутренние данные виджета.

Ресурсы array (указательный метод)

Если Вы получаете ресурсы массива, используя метод указателя:

- Аргумент value в PtSetArg() является адресом указателя соответствующего типа языка C (первый из двух типов языка C, данный в таблице "Новые ресурсы").
- len является адресом указателя на данный второй тип языка C.

Когда вызывается PtGetResources():

- Указатель, задаваемый в value, устанавливается указывающим на начало массива во внутреннем хранилище виджета.
- Указатель, задаваемый в len, устанавливается указывающим на счётчик элементов массива во внутреннем хранилище виджета.

Ресурсы pointer (указательный метод)

Если Вы получаете ресурсы указателя, используя метод указателя:

- Аргумент value в PtGetArg() является адресом указателя соответствующего типа языка C.
- len не используется.

Когда вызывается PtGetResources(), задаваемый указатель устанавливается указывающим на те же данные, что и внутренний указатель виджета. Данные являются внешними по отношению к виджету; Вы можете модифицировать их в зависимости от ресурса.

Ресурсы link (указательный метод)

Если Вы получаете ресурсы связанного списка, используя метод указателя:

- Аргумент value в PtSetArg() является адресом указателя на списочную структуру типа PtLinkedList_t. Эта структура содержит по меньшей мере:
struct Pt_linked_list *next указатель на следующий пункт списка;
char data[1] адрес данных, хранящихся в списке;
- len не используется.

Когда вызывается PtGetResources(), указатель, заданный в value, устанавливается указывающим на первый узел связанного списка во внутренних данных виджета.

☞ Если Вы получаете ресурс ответной реакции, аргумент value в PtSetArg() является адресом указателя на структуру типа PtCallbackList_t. Для получения более полной информации см. раздел "Проверка ответных реакций" в главе "Управление виджетами в исходном коде приложения".

Ресурсы struct (указательный метод)

Если Вы получаете ресурсы структуры, используя метод указателя:

- Аргумент value в PtSetArg() является адресом указателя соответствующего типа языка C.
- len не используется.

Когда вызывается функция PtGetResources(), заданный указатель для ресурса структуры устанавливается указывающим на внутреннее хранилище виджета.

Ресурсы boolean (указательный метод)

Если Вы получаете булевский ресурс, используя метод указателя:

- Аргумент value в PtSetArg() является указателем на int.
- len не используется.

Когда вызывается функция PtGetResources(), int устанавливается в 1, если булевское значение "истина", или в 0, если "ложь". Например, чтобы получить значение ресурса Pt_ARG_CURSOR_OVERRIDE в виджете типа PtContainer:

```
PtArg_t arg;
int bool_value;

PtSetArg( &arg[0], Pt_ARG_CURSOR_OVERRIDE, &bool_value, 0 );
PtGetResources( ABW_container, 1, arg );

if ( bool_value ) {
    /* Курсор контейнера перекрывает курсор своих детей. */
}
```

Вызов функции PtGetResources()

Используйте функцию PtGetResources(), чтобы получить значения каждого ресурса, определённого в списке аргументов:

```
int PtGetResources(PtWidget_t *widget, int n_args, PtArg_t *args);
```

Аргументы этой функции являются идентификатором виджета, числом входов в список аргументов и сам по себе список аргументов.

PtGetResources() возвращает 0 в случае успеха или -1, если случается ошибка. Возврат кода, равного -1, может указывать на то, что Вы пытаетесь получить значение ресурса, который для виджета не определён.

Получение одного ресурса

Если Вы получаете значение одного ресурса, проще использовать вместо функции PtGetResources() функцию PtGetResource(). Для функции PtGetResource() Вам нет необходимости устанавливать список аргументов. Аргументами PtGetResource() являются:

```
int PtGetResource (PtWidget_t *widget, long type, long value, long len);
```

widget – это указатель на виджет, ресурс которого мы получаем. Другие аргументы устанавливаются точно так же, как и в PtSetArg(), когда с использованием указательного метода получаете более одного ресурса.

Вот пример получения одного ресурса функцией PtGetResources() и указательного метода:

```
unsigned short *width;  
PtArg_t arg;  
  
PtSetArg( &arg, Pt_ARG_BEVEL_WIDTH, &width, 0 );  
PtGetResources( widget, 1, &arg );
```

При использовании функции PtGetResource() код имеет такой вид:

```
unsigned short *width;  
  
PtGetResource( widget, Pt_ARG_BEVEL_WIDTH, &width, 0 );
```



PtGetResource() возвращает указатель непосредственно во внутреннюю память виджета. Не пытайтесь модифицировать ресурс, напрямую используя этот указатель. Такая модификация не приведёт к ожидаемому эффекту и вызовет ошибки поведения виджета. **Никогда не освобождайте указатель** – результатом этого наверняка будет ошибка памяти или другая сходная ошибка. Использование указателя const поможет избежать таких проблем.

Изменение состояния виджета может сделать указатель потерявшим силу, используйте его сразу же.

Глава 11. Управление виджетами в исходном коде приложения

Мы рекомендуем Вам создавать пользовательский интерфейс Вашего приложения в PhAB – это проще, чем делать это в Вашем коде. Однако если интерфейс динамический, Вы, возможно, создаёте часть интерфейса "на лету".

Эта глава включает:

- Создание виджетов
- Задание порядка виджетов
- Ответные реакции
- Обработчики событий
- Стили виджетов

Создание виджетов

Создание виджета в Вашем приложении требует немного больше работы, чем создание его в PhAB. Это потому, что PhAB заботится о куче физических атрибутов для Вас, включая размеры, местоположение и прочая. Если Вы создаёте виджет в своём коде, Вы должны будете сами установить эти ресурсы. Чтобы создать виджет из своего кода, вызовите функцию `PtCreateWidget()`. Её синтаксис следующий:

```
PtWidget_t *PtCreateWidget(
    PtWidgetClassRef_t *class,
    PtWidget_t *parent,
    unsigned n_args,
    PtArg_t *args );
```

Её аргументами являются:

- class* Тип создаваемого виджета (напр., `PtButton`)
- parent* Родитель нового виджета. Если это `Pt_DEFAULT_PARENT`, новый виджет становится потомком родителя, принимаемого по умолчанию, которым является самый последний созданный виджет контейнерного класса. Если *parent* равно `Pt_NO_PARENT`, виджет не имеет родителя.
- n_args* Число элементов массива *args*
- args* Массив структур `PtArg_t`, который хранит Ваши установки для ресурсов виджета. Эти установки аналогичны тем, что используются для `PtSetResources()`; см. главу "Управление ресурсами в исходном коде приложения".

Вы можете задать принимаемого по умолчанию родителя (используемого, если аргумент *parent* в вызове функции `PtCreateWidget()` равен `Pt_DEFAULT_PARENT`), вызвав функцию `PtSetParentWidget()`. Чтобы назначить для виджета другой контейнер, вызовите функцию `PtReparentWidget()`.

Вот несколько заметок относительно виджетов, создаваемых в исходном коде приложения:

- Виджет не реализуется, пока не реализуется контейнерный виджет. Если контейнер уже реализован, Вы можете вызвать функцию `PtRealizeWidget()`, чтобы реализовать новый виджет.
- Если Вы создали виджет в модуле PhAB и затем уничтожили модуль, виджет тоже будет уничтожен. В следующий раз, когда модуль будет создан, он появится таким, каким он был задан в PhAB.
- если Вы сохранили в виджете глобальный указатель, убедитесь, что при уничтожении виджета Вы сбросили указатель в `NULL`. Это легко сделать в ответной реакции `Pt_CB_DESTROYED` виджета. Ошибка, заключающаяся в несбрасывании глобального указателя (и проверки его перед использованием), является часто встречающимся источником проблем с виджетами, созданными через программный код.

Задание порядка виджетов

Порядок, в котором виджетам даётся фокус, зависит от порядка, в котором они были созданы или от порядка виджетов, заданного в PhAB (см. "Задание порядка виджетов" в главе "Создание виджетов в PhAB"). Самым задним виджетом является первый в установленном порядке перехода, самый передний является последним.

Если Вы создаёте виджеты программно, Вы можете создавать их в том порядке, в котором Вы хотите передавать им фокус, или же Вы можете использовать для изменения порядка следующие функции:

- `PtWidgetInsert()` Вставить виджет в иерархию семейства виджетов
- `PtWidgetToBack()` Переместить виджет за спину всех своих братьев
- `PtWidgetToFront()` Переместить виджет впереди всех своих братьев

С другой стороны, Вы можете использовать ответную реакцию виджета `Pt_CB_LOST_FOCUS` (определённую для `PtBasic`), чтобы переписать порядок перехода путём передачи фокуса другому виджету.

В ответной реакции потери фокуса используйте функцию `PtContainerGiveFocus()`, чтобы передать фокус нужному виджету, и возвращайте для ответной реакции `Pt_END`, чтобы воспрепятствовать передаче фокуса первоначальной цели изменения фокуса.

☞ Ответная реакция `Pt_CB_LOST_FOCUS` вызывается во второй раз, когда фокус перемещается от виджета к новой цели. Чтобы исключить бесконечную петлю, используйте статическую переменную для указания того, что эта ответная реакция уже перенаправила фокус.

Работа с семейством виджетов

Для работы с иерархией семейства виджетов могут использоваться следующие функции, и они могут быть полезны при установке порядка передачи фокуса:

<code>PtChildType</code>	Определяет взаимосвязь между двумя виджетами
<code>PtFindDisjoint()</code>	Возвращает ближайший рассоединённый родительский виджет
<code>PtFindFocusChild()</code>	Находит ближайший фокусируемый сыновний виджет
<code>PtFindGuardian()</code>	Находит виджет, отвечающий за действие другого виджета
<code>PtGetParent()</code>	Находит ближайший родительский виджет, совпадающий с заданным классом
<code>PtGetParentWidget()</code>	Возвращает текущий действующий по умолчанию родительский виджет
<code>PtNextTopLevelWidget()</code>	Получает указатель на следующий виджет верхнего уровня
<code>PtValidParent()</code>	Идентифицирует действующего родителя виджета
<code>PtWidgetBrotherBehind()</code>	Получает брата позади виджета
<code>PtWidgetBrotherInFront()</code>	Получает брата впереди виджета
<code>PtWidgetChildBack()</code>	Получает сына, самого заднего в контейнере
<code>PtWidgetChildFront()</code>	Получает сына, самого переднего в контейнере
<code>PtWidgetFamily()</code>	Проходит иерархию виджета задом наперёд
<code>PtWidgetParent()</code>	Получает родителя виджета
<code>PtWidgetSkip()</code>	Перескакивает к виджету в следующей иерархии
<code>PtWidgetTree()</code>	Проходит дерево виджетов спереди назад
<code>PtWidgetTreeTraverse()</code>	Проходит иерархию семейства виджетов спереди назад

Ответные реакции

Вы можете добавлять и удалять ответные реакции в Вашем программном коде так же, как и через PhAB – только обратите внимание на различия между двумя типами!

Добавление ответных реакций

Приложение регистрирует ответные реакции, манипулируя ресурсами ответной реакции виджета. Для этих ресурсов в классах виджетов Photon'a используется соглашение по именованию: они все начинаются с `Pt_CB_`.

Ответные реакции могут быть добавлены в список ответных реакций, хранимых для этих ресурсов, использованием функции `PtAddCallbacks()` для добавления в список нескольких функций ответных реакций или функции `PtAddCallback()` для добавления только одной. В любом случае первыми двумя аргументами функции являются виджет и имя ресурса ответной реакции, которая добавляется. Остальные аргументы зависят от того, какая функция используется.

В функции `PtAddCallbacks()` третьим аргументом является массив записей ответных реакций. Каждая запись содержит указатель на функцию ответной реакции и указатель на данные присоединённого клиента, которые будут передаваться функции ответной реакции при её вызове. Каждая из этих записей ответных реакций скопирована во внутренний список ответных реакций виджета.

Например, мы можем захотеть, чтобы, когда пользователь выбрал кнопку (т.е. нажал её), приложение выполнило некое действие. Класс `PtButton` виджетов предоставляет ресурс ответной реакции `Pt_CB_ACTIVATE` для уведомления приложения, когда кнопка нажата. Чтобы создать виджет и присоединить к этому ресурсу ответной реакции функцию ответной реакции, мы используем код такого вида:

```
{
    PtWidget_t *button;
    int push_button_cb( PtWidget_t *, void *, PtCallbackInfo_t *);
    PtCallback_t callbacks[] = { {push_button_cb, NULL} };

    ...

    button = PtCreateWidget(PtButton, window, 0, NULL);
    PtAddCallbacks(button, Pt_CB_ACTIVATE, callbacks, 1);
}
```

где `push_button_cb` – имя функции приложения, которая должна вызываться, когда пользователь нажимает кнопку. Для определения списка ответных реакций используется структура `PtCallback_t`; более подробно см. "Справочник виджетов Photon'a".

Когда в список ответных реакций добавляется только одна функция ответной реакции (как в этом случае), проще использовать `PtAddCallback()`. Эта функция берёт указатель на функцию ответной реакции как третий аргумент и указатель на данные клиента как последний аргумент. Вышеприведенный фрагмент кода может быть записан более кратко таким образом:

```
{
    PtWidget_t *button;
    int push_button_cb( PtWidget_t *, void *, PtCallbackInfo_t *);
    button = PtCreateWidget(PtButton, window, 0, NULL);
    PtAddCallback(button, Pt_CB_ACTIVATE, push_button_cb, NULL);
}
```

Вы можете также передавать массив записей ответной реакции как значение для ресурса ответной реакции, когда используется список аргументов в связке с функцией `PtCreateWidget()` или `PtSetResources()`. Поскольку список ответных реакций является массивом, Вы должны задать базовый адрес массива как третий аргумент функции `PtSetArg()` и число элементов как последний аргумент. В этом случае записи ответной реакции добавляются к текущему списку ответной

реакции, если он имеется. Это даёт нам другой способ задания ответной реакции для вышеприведенного примера:

```
{
    PtArg_t arg[5];
    int push_button_cb( PtWidget_t *, void *, PtCallbackInfo_t *);
    PtCallback_t callbacks[] = { {push_button_cb, NULL} };
    ...
    PtSetArg(&args[0], Pt_CB_ACTIVATE, callbacks, 1);
    PtCreateWidget(PtButton, window, 1, arg);
}
```

Каждый из этих методов имеет свои достоинства. Использование PtAddCallback(), конечно, простое. Использование функции PtAddCallbacks() более эффективно, когда имеется несколько ответных реакций. Использование функции PtSetArg() и передача результата в PtCreateWidget() позволяет автоматически осуществлять создание виджета и прикрепить список ответных реакций.

Вызов ответной реакции

Функция ответной реакции вызывается со следующими параметрами:

Pt_Widget_t *widget	Виджет, послуживший причиной вызова функции ответной реакции, т.е. то, на чём произошло действие.
void *client_data	Специфические для данного приложения данные, которые были присоединены к ответной реакции, когда та была зарегистрирована в виджете.

☞ Клиентские данные, которые передаются ответной реакции, добавляемой из Вашего программного кода, не являются такими же, как данные arinfo, передаваемые в ответные реакции PhAB.

PtCallbackInfo_t *call_data	Указатель на структуру PtCallbackInfo_t (см. "Справочник виджетов Photon'a"), которая хранит данные, специфические для этого вызова ответной реакции. Она зависит от причин, по которым ответная реакция была вызвана, и может иметь данные, специфические для поведения ответной реакции. Структура PtCallbackInfo_t определена как:
-----------------------------	--

```
typedef struct Pt_callback_info {
    unsigned long    reason;
    unsigned long    reason_subtype;
    PhEvent_t        *event;
    void              *cbdata;
} PtCallbackInfo_t;
```

Элементы структуры PtCallbackInfo_t имеют следующий смысл:

- *reason* – указывает причину вызова ответной реакции; обычно это устанавливается на имя ресурса ответной реакции, чей список ответных реакций был вызван.
- *reason_subtype* – указывает конкретный тип ответной реакции, связанный с reason; для большинства ответных реакций это значение равно 0.
- *event* – указатель на структуру PhEvent_t (см. "Справочник библиотечных функций Photon'a"), которая описывает событие Photon'a, послужившее причиной вызова ответной реакции.
- *cbdata* – вызывает данные, которые являются специфическими для ресурса ответной реакции, послужившей причиной вызова функции ответной реакции.

Для получения более полной информации см. описание ответных реакций, определённых для каждого виджета, в "Справочнике виджетов".

Удаление ответных реакций

Вы можете удалить одну или более ответных реакций из списка ответных реакций, связанных с ресурсом виджета, используя функции `PtRemoveCallbacks()` и `PtRemoveCallback()`.



Не пытайтесь удалять ответные реакции, которые были добавлены через PhAB; результатом может стать непредсказуемое поведение.

Функция `PtRemoveCallbacks()` берёт массив записей ответных реакций как аргумент и удаляет все ответные реакции, заданные ей в списке ответных реакций. Функция `PtRemoveCallback()` удаляет только одну функцию ответной реакции из списка ответных реакций. Обе функции берут виджет как первый аргумент и ресурс виджета как второй аргумент.

Чтобы удалить ответную реакцию с кнопки, созданной нами выше, мы должны сделать следующее:

```
int push_button_cb( PtWidget_t *, void *, PtCallbackInfo_t *);
PtCallback_t callbacks[] = { {push_button_cb, NULL} };
PtRemoveCallbacks(button, Pt_CB_ACTIVATE, callbacks, 1);
```

или так:

```
int push_button_cb( PtWidget_t *, void *, PtCallbackInfo_t *);
PtRemoveCallback(button, Pt_CB_ACTIVATE, push_button_cb,
```

[Прим.пер. – так в тексте, без последнего параметра. Вероятно, должен быть равен 1].

Оба – указатель на функции ответной реакции и указатель на данные клиента – важны при удалении ответных реакций. Только первый элемент списка ответных реакций, который имеет одновременно ***и*** ту же функцию ответной реакции, ***и*** тот же указатель на данные клиента, будет удалён из списка ответных реакций.

Просмотр ответных реакций

Вы можете просмотреть список ответных реакций, чтобы получить значение соответствующего ресурса списка ответных реакций. Тип значения, которое Вы получаете из ресурса списка ответных реакций, отличается от значения, использованного для установки ресурса. Несмотря на то, что этот ресурс установлен на массив записей ответных реакций, значение, полученное при получении ресурса, является указателем на список записей ответных реакций. Типом этого списка является `PtCallbackList_t`. Каждый элемент списка содержит член `cb` (т.е. запись ответной реакции) и *следующий* указатель (который указывает на следующий элемент списка).

Следующий пример показывает, как Вы можете пройти по списку ответных реакций `Pt_CB_ACTIVATE` для виджета, чтобы найти все экземпляры конкретной функции ответной реакции `cb`:

```
...
PtCallbackList_t *cl;

PtGetResources(widget, Pt_CB_ACTIVATE, &cl, 0);
for ( ; cl; cl = cl->next )
{
    if ( cl->cb.func == cb )
        break;
}
```

Обработчики событий

Вы можете добавлять и удалять обработчики событий (необработанные или отфильтрованные ответные реакции) в программном коде Вашего приложения, так же как и в PhAB – однако, имеются несколько отличий между двумя типами.

☞ Описание необработанных и отфильтрованных ответных реакций, и то, как это используется, см. в разделе "Обработчики событий – необработанные и отфильтрованные ответные реакции" в главе "События".

Для получения более полной информации по добавлению обработчиков событий в PhAB см. раздел "Обработчики событий – необработанные и отфильтрованные ответные реакции" в главе "Редактирование ресурсов и ответных реакций в PhAB".

Добавление обработчиков событий

Как и в случае ответных реакций, Вы так же можете установить или просмотреть обработчики событий, чтобы выполнить установку или получить данные, в ресурсе обработчика событий. Следующие ресурсы в PtWidget позволяют Вам задавать обработчики для событий Photon'a:

- Pt_CB_FILTER
- Pt_CB_RAW

Для получения более полной информации по этим ресурсам ответных реакций см. "Справочник виджетов Photon'a".

Операция установки требует массива записей обработчиков событий типа PtRawCallback_t. Они похожи на записи ответных реакций, обсуждавшиеся выше, и имеют поля event_mask, event_f и data.

Поле event_mask – это маска типов событий Photon'a (см. описание PhEvent_t в "Справочнике библиотечных функций Photon'a"), указывающая, какие события будут являться причиной вызова функции ответной реакции [Прим. пер. Наверное, это неточность: не функции ответной реакции, а функции обработчика]. Члены event_f и data – это функция обработчика события и данные клиента соответственно.

☞ Если Вы добавляете обработчик событий реализованному виджету и регион виджета нечувствителен к одному или более типов событий, содержащихся в маске событий, то регион делается чувствительным к ним.

Если Вы добавляете обработчик событий *перед тем*, как реализовать виджет, Вы должны приспособить чувствительность региона сами, после того как виджет будет реализован. См. функцию PhRegionChange() в "Справочнике библиотечных функций Photon'a".

Операция получения данных получает список PtRawCallbackList* записей обработчиков событий. Как и в случае списка ответных реакций, список содержит два члена: next и cb. Член cb – это запись обработчика события.

Вы можете добавить обработчики событий Pt_CB_RAW, используя функции PtAddEventHandler() или PtAddEventHandlers(). Вы можете добавить обработчики событий Pt_CB_FILTER, используя функции PtAddFilterCallback() или PtAddFilterCallbacks(). Аргументами PtAddEventHandler() и PtAddFilterCallback() являются:

<i>widget</i>	Виджет, к которому должен быть добавлен обработчик события
<i>event_mask</i>	Маска событий определяет, какие события будут приводить к вызову обработчика событий
<i>event_f</i>	Функция обработки события
<i>data</i>	Указатель, передаваемый обработчику событий и указывающий на клиентские данные

Аргументами функций `PtAddEventHandlers()` и `PtAddFilterCallbacks()` являются:

<i>widget</i>	Виджет, к которому должны быть добавлены обработчики событий
<i>handlers</i>	Массив записей обработчиков событий
<i>nhandlers</i>	Число обработчиков событий, задаваемых в массиве

Удаление обработчиков событий

Вы можете удалить обработчики событий `Pt_CB_RAW`, вызвав функции `PtRemoveEventHandler()` или `PtRemoveEventHandlers()`. Вы можете удалить обработчики событий `Pt_CB_FILTER`, вызвав функции `Pt_RemoveFilterCallback()` или `Pt_RemoveFilterCallbacks()`.



Не удаляйте обработчики событий, которые были добавлены через PhAB; результатом может стать непредсказуемое поведение.

Параметрами функций `PtRemoveEventHandler()` и `PtRemoveFilterCallback()` являются:

<i>widget</i>	Виджет, с которого удаляется обработчик событий
<i>event_mask</i>	Маска событий, задающая события, к которым чувствителен обработчик
<i>event_f</i>	Функция обработки события
<i>data</i>	Данные клиента, присоединённые к обработчику

Это выполняет поиск обработчика событий с такой сигнатурой – т.е. идентичными *event_mask*, *data* и *event_f* – в виджете и удалении такового, если он найден.

Параметрами функций `PtRemoveEventHandlers()` и `PtRemoveFilterCallbacks()` являются:

<i>widget</i>	Виджет, с которого удаляются обработчики событий
<i>handlers</i>	Массив записей обработчиков событий
<i>nhandlers</i>	Число обработчиков событий, определённых в массиве

Как и для функций `PtRemoveEventHandler()` и `PtRemoveFilterCallback()`, обработчик событий удаляется только тогда, когда он имеет в точности ту же сигнатуру, что и один из обработчиков событий, заданных в массиве записей обработчиков событий.

Запуск обработчика событий

При запуске обработчики событий получают те же аргументы, что и функции ответных реакций, т.е. параметрами являются:

- виджет, получивший событие (*widget*)
- данные клиента, присоединённые к обработчику событий (*client_data*)
- ☞ Данные клиента, передаваемые обработчику событий, не являются такими, как и данные *arinfo*, передаваемые обработчику событий, добавленному через PhAB.
- информация ответной реакции, связанная с конкретным событием (*info*).

Обработчики событий возвращают целое значение, которое обработчик события должен использовать для указания на то, должна ли или не должна производиться дальнейшая обработка события. Если обработчик событий возвращает значение `Pt_END`, это указывает, что дальнейшая обработка события `Photon` не выполняется, и событие исчерпано.

Член *event* параметра *info* содержит указатель на события, послужившие причиной запуска обработчика событий. Вы должны проверить для этого события член *type*, чтобы определиться, как поступить с этим событием. Это будет один из типов событий, определённых в *event_mask*, заданной при добавлении обработчика событий к виджету.

Для получения данных, присоединённых к конкретному событию, вызывается функция `PhGetData()` с указателем на событие в качестве параметра. Эта функция вернёт указатель на структуру с данными, специфическими для данного конкретного типа события. Этот тип структуры зависит от типа события.

Стили виджетов

Стили классов виджетов позволяют Вам на лету подстраивать под себя или модифицировать внешний вид виджетов, их размеры и поведение. Они также обеспечивают различный внешний вид одного и того же типа виджета, существующие одновременно. В сущности, стиль класса виджета представляет из себя набор методов и данных, определяющих внешний вид и реакции экземпляров класса виджета.

Каждый класс виджета имеет принимаемый по умолчанию стиль, но Вы можете в любой момент добавить или модифицировать неограниченное количество дополнительных стилей. Вы можете даже модифицировать принимаемый по умолчанию для данного класса стиль, изменив внешний вид и реакцию любых экземпляров этого класса, которые используют принимаемый по умолчанию стиль.

Каждый экземпляр виджета может ссылаться на определённый стиль, обеспечиваемый его классом. Вы можете изменить стиль, которым какой-либо виджет сможет воспользоваться в любой момент, когда это Вам понадобится.

Каждый стиль имеет набор членов, включающих имя стиля и функции, которые заменяют или расширяют какие-либо методы класса виджета. Методы являются функциями уровня класса, определяющими, как виджет инициализирует себя, прорисовывает себя, вычисляет свои размеры и прочая. Для получения более полной информации о методах см. руководство "Построение подстраиваемых виджетов". Члены стиля определены следующими декларациями:

Pt_STYLE_DRAW	Адрес функции, вызываемой всякий раз виджетом, использующим этот стиль, когда ему требуется перерисовать себя.
Pt_STYLE_EXTENT или Pt_STYLE_SIZING	Адрес функции, вызываемой всякий раз виджетом, использующим этот стиль, когда происходит перемещение, изменение размеров или модификация такого рода, которая может потребовать перемещения виджета или изменения его размеров (изменение в данных виджета). Эта функция отвечает за установку размеров виджета в соответствующие значения.
Pt_STYLE_ACTIVATE	Адрес функции, вызываемой всякий раз виджетом, создаваемым со стилем, принятым по умолчанию, и всякий раз, когда стиль виджета изменяется с одного стиля на другой. Эта функция является тем местом, где размещается манипулирование плоскостями управления виджета, дополнения к ответным реакциям или установки ресурсов (для перезаписи принятых для виджета по умолчанию).
Pt_STYLE_CALC_BORDER	Адрес функции, отвечающей за оповещение как много пространства требуется для представления отделки и границ краёв виджета.
Pt_STYLE_CALC_OPAQUE	Адрес функции, отвечающей за вычисление списка "черепицы", представляющий затенённые области виджета. Этот список используется для определения того, что должно быть повреждено под этим виджетом, когда он модифицируется.
Pt_STYLE_DEACTIVATE	Адрес функции, вызываемой всякий раз виджетом, использующим этот стиль, когда виджет уничтожается либо переключается на другой стиль.
Pt_STYLE_NAME	Имя стиля
Pt_STYLE_DATA	Указатель на произвольный блок данных для использования стилем.

Более детально описание этих членов см. в описании библиотечной функции Photon'a PtSetStyleMember().

Следующие функции позволяют Вам создавать и манипулировать стилями класса виджета:

PtAddClassStyle()	Добавляет стиль к классу виджета
PtCreateClassStyle()	Создаёт стиль класса
PtDupClassStyle()	Получает копию стиля класса виджета
PtFindClassStyle()	Отыскивает стиль с заданным именем
PtGetStyleMember()	Получает член стиля
PtGetWidgetStyle()	Получает стиль, который виджет использует в текущий момент
PtSetStyleMember()	Устанавливает член стиля
PtSetStyleMembers()	Устанавливает множество членов стиля из списка аргументов переменной длины
PtSetWidgetStyle()	Устанавливает текущий стиль для виджета

Некоторые из этих функций требуют или возвращают указатель на структуру PtWidgetClassStyle_t. Не обращайтесь к членам этой структуры напрямую, а используйте для этого функцию PtGetStyleMember().

☞ Вы можете также установить стиль экземпляра виджета, установив его ресурс Pt_ARG_STYLE (см. описание виджета PtBasic в книге "Справочник виджетов"). Установка этого ресурса имеет тот же эффект, что и вызов функции PtSetWidgetStyle().

Этот пример создаёт стиль, названный blue, и несколько кнопок. Заметьте, что Ваши виджеты могут использовать стиль до того, как Вы добавили стиль к классу или даже перед тем, как Вы создали стиль. Когда Вы создаёте стиль и добавляете его, все виджеты, использующие стиль, немедленно обновляются.

```
#include <Pt.h>

PtWidget_t *win, *but;
PtWidgetClassStyle_t *b;

void blue_draw (PtWidget_t *widget, PhTile_t *damage)
{
    /* Это функция прорисовки для стиля blue.
       Она рисует голубой прямоугольник (без надписи)
       для виджета.
    */

    PgSetFillColor( Pg_BLUE);
    PgDrawRect( PtWidgetExtent (widget, NULL), Pg_DRAW_FILL);
}

int use_blue_style( PtWidget_t *widget, void *data, PtCallbackInfo_t *cbinfo)
{
    /* Эта ответная реакция устанавливает текущий стиль для данного
       экземпляра виджета. Если Вы не прикрепили стиль blue
       к классу, никаких изменений во внешнем виде виджета нет.
    */

    PtSetWidgetStyle (widget, "blue");
    return Pt_CONTINUE;
}

int attach_blue_style( PtWidget_t *widget, void *data,
                      PtCallbackInfo_t *cbinfo)
{
    /* Эта ответная реакция добавляет стиль к классу виджета.
       Если Вы щёлкнули на одной из кнопок "использование стиля blue",
       изменится стиль всех кнопок.
    */

    PtAddClassStyle (PtButton, b);
    return Pt_CONTINUE;
}
```

```
int main()
{
    PhArea_t area = {{0,50},{100,100}};
    PtArg_t argt[10];
    PtStyleMethods_t meth;
    PtCallback_t cb = {use_blue_style, NULL};
    PtCallback_t cb2 = {attach_blue_style, NULL};
    int unsigned n;

    /* Инициализация методов для стиля. */
    meth.method_index = Pt_STYLE_DRAW;
    meth.func = blue_draw;

    PtInit(NULL);

    /* Создание окна. */
    PtSetArg (&argt[0], Pt_ARG_DIM, &area.size, 0);
    win = PtCreateWidget (PtWindow, NULL, 1, argt);

    /* Создание нескольких кнопок. Когда Вы щёлкаете по одной из этих
       кнопок, ответная реакция делает экземпляр виджета
       использующим стиль blue. */
    n = 0;
    PtSetArg (&argt[n++], Pt_ARG_TEXT_STRING, "Use blue style", 0);
    PtSetArg (&argt[n++], Pt_CB_ACTIVATE, &cb, 1);
    but = PtCreateWidget (PtButton, NULL, n, argt);

    PtSetArg (&argt[0], Pt_ARG_TEXT_STRING, "Use blue style also", 0);
    PtSetArg (&argt[n++], Pt_ARG_POS, &area.pos, 0);
    but = PtCreateWidget (PtButton, NULL, n, argt);

    /* Создание другой кнопки. Когда Вы щёлкаете на ней,
       ответная реакция прикрепляет стиль blue к классу виджета. */
    n = 0;
    PtSetArg (&argt[n++], Pt_ARG_TEXT_STRING, "Attach blue style", 0);
    PtSetArg (&argt[n++], Pt_CB_ACTIVATE, &cb2, 1);
    PtSetArg (&argt[n++], Pt_ARG_POS, &area.pos, 0);
    area.pos.y = 85;
    but = PtCreateWidget (PtButton, NULL, n, argt);

    /* Копирование принимаемого по умолчанию стиля, чтобы создать стиль blue.
       Замена рисуемого члена новым стилем. */
    b = PtDupClassStyle (PtButton, NULL, "blue");
    PtSetClassStyleMethods (b,1,&meth);

    PtRealizeWidget (win);
    PtMainLoop();

    return EXIT_SUCCESS;
}
```

Глава 12. Поверхности управления

Что такое поверхности управления?

Поверхности управления – это геометрические области внутри виджета, которые могут позиционировать, изменять размеры и перерисовывать сами себя. Кроме того, они могут определять своё собственное поведение. Они делают всё это через ответные реакции и флаги управления событиями, предоставляемыми при создании поверхности.

Поверхности управления позволяют Вам переопределять поведение для любой области внутри рисуемого пространства виджета. Кроме того, они могут перерисовывать себя, равно как и пересчитывать свою собственную геометрию. Концептуально они могут рассматриваться как облегчённые "виджеты внутри виджетов".

Например, рассмотрим линейку протяжки (scroll bar). Вы получаете различные действия в зависимости от того, где Вы щёлкнете на ней: кнопки стрелок выполняют шаг вверх или вниз; щелчок на жёлобе выполнит пролистывание страницы вверх или вниз; перетаскивание каретки выполняет прокрутку при её перемещении. Виджет PtScrollbar выполнен как одиночный виджет с несколькими поверхностями управления на нём. Вы можете также использовать плоскости управления для:

- эмулирования клавиатуры, напр., добавление Shift-lock к клавише Shift;
- панелей нажимных кнопок;
- и прочего.

Важно отметить, что поверхности управления – это свойство виджета; для своего существования им требуется виджет. Вместе с тем виджет может обладать любым числом поверхностей управления, делая возможным при использовании всего одного виджета (напр., PtWindow) обеспечивать целый пользовательский интерфейс на ничтожном размере данных при исполнении (8% является разумной верхней границей) как альтернативу обеспечению несколькими виджетами для использования в пользовательском интерфейсе.

Ограничения

Для поверхностей управления существует несколько ограничений:

- Библиотека виджетов предоставляет сервис для виджетов, который не может, из соображений экономии, обеспечиваться для поверхностей управления. Например, виджеты обладают понятием затенённости, используемым при прорисовке для снижения мерцания. Поверхности управления просто прорисовываются от тыльной к передней без какой-либо заботы о затенении.
- Поверхности управления не могут содержать других поверхностей управления и не включают понятие фокуса.
- Поверхности управления являются крайне необработанными элементами и могут обеспечивать только то поведение, которое Вы заложили при их создании. Нетрудно реализовать кнопку как поверхность управления, но построение виджета PtMultitext как такую поверхность потребует уже больших усилий.

Привязка действий к поверхностям управления

Вы можете привязать поверхности управления к любым предопределённым действиям виджета или к действиям, определённым пользователем.

Типами поверхностей управления являются:

Обычные поверхности	Позволяют Вам определять маску событий и функции ответных реакций для поверхности управления
Активные поверхности	Позволяют Вам автоматически связывать поверхность управления с одним из предопределённых действий виджета

Ссылка на поверхности управления

Вы можете ссылаться на поверхности управления через:

- указатель на структуру поверхности управления (`PtSurface_t*`)
- числовой идентификатор (16-битовые `PtSurfaceId_t`).

Этот идентификатор уникальным образом идентифицирует поверхность управления внутри его соотнесённого виджета. Действительными значениями для идентификатора поверхности являются значения в диапазоне от 1 до 255 включительно.

Хотя метод указателя действует напрямую и потому более быстр, он не настолько безопасен, как метод идентификатора. Чтобы понять почему, обсудим, как поверхности управления организованы и хранятся в памяти.

В отличие от иерархии виджетов, выполненной как связанный список, поверхности управления хранятся как массив структур поверхностей (`PtSurface_t`). Основными причинами для хранения таким способом являются:

- Массив обеспечивает быстрое прохождение в обоих направлениях (что требуется, поскольку прорисовка обрабатывается от тыла к переднему плану, а события обрабатываются от переднего плана к заданному).
- Массив снижает запросы на память в расчёте на одну поверхность. Для удовлетворения требования быстрого прохождения должен быть использован дважды связанный список, что существенно увеличивает требуемый объём памяти.
- Вам, вероятно, не очень часто понадобится добавлять или удалять поверхности управления, так что использование массива не станет причиной большого ухудшения производительности.

Поскольку Вы физически перемещаете поверхности управления по кругу в стековом порядке, их размещение в массиве изменяется, в результате чего меняются их адреса в памяти. Кроме того, при добавлении или удалении Вами поверхностей управления к виджету, массив необходимо переразмещать, что также станет причиной того, что сам массив в целом будет перемещаться в памяти. Со всеми этими возможностями по перемещению в памяти, цифровые идентификаторы являются единственным реальным способом локализации поверхности.

Если Вы в достаточной степени уверены, что конфигурация поверхностей виджета не будет изменяться, то метод указателя является безопасным (и более быстрым, поскольку метод идентификатора требует исполнения линейной петли обработки в массиве поверхности).

API поверхностей управления

Функции, приведенные ниже, описаны в "Справочнике библиотечных функций Photon'a").

Создание и уничтожение поверхностей управления

Следующие функции создают и уничтожают поверхности управления:

PtCreateActionSurface()	Создаёт поверхность управления внутри виджета, связанную с действием виджета
PtCreateSurface()	Создаёт обычную поверхность управления внутри виджета
PtDestroyAllSurface()	Уничтожает все поверхности управления виджета
PtDestroySurface()	Уничтожает поверхность управления
PtDestroySurfaceById()	Уничтожает поверхность управления с заданным идентификатором

Нахождение идентификаторов для поверхностей управления

Чтобы найти идентификатор поверхности и действия, используйте эти функции:

PtSurfaceActionId()	Получает идентификатор действия для поверхности
PtSurfaceId()	Получает идентификатор поверхности управления

Вычисление геометрии для поверхностей управления

Вы должны предусмотреть функцию, которая вычисляет геометрию управляющей поверхности. Поверхности управления опрашиваются для вычисления их геометрии дважды, когда виджет, являющийся их владельцем, опрашивается для вычисления его геометрии:

- непосредственно перед вычислением геометрии виджета (что позволяет виджету изменить свои размеры в соответствии с требованиями своих поверхностей, если виджет заботится об этом – и некоторые виджеты так делают это);
- сразу после (позволяя поверхностям позиционировать и задать свои размеры в соответствии с размерами виджета).

Аргумент почтового сообщения, передаваемый функции геометрии, говорит Вам, какой случай обрабатывается.

Поверхность может также вычислять свою геометрию на основании геометрии других поверхностей. Используя функции PtCalcSurface() или PtCalcSurfaceById(), вы можете гарантировать, что поверхности, которыми Вы интересуетесь, вычислили свою геометрию до того, как их рассматривают.

Текущая запись геометрии поверхности – это просто вопрос прямой модификации массива указателя поверхностей. Убедитесь, что Вы знаете, как этот массив организован, перед тем как обрабатывать его. Эта организация подробно рассмотрена в документации к функции PtCreateSurface().

Эти функции имеют дело с геометрией поверхностей управления:

PtCalcSurface()	Вынуждает поверхность вычислить свою геометрию
PtCalcSurfaceByAction()	Вынуждает все поверхности, связанные с действием, вычислить свою геометрию
PtCalcSurfaceById()	Вынуждает поверхность управления с заданным идентификатором вычислить свою геометрию
PtSurfaceCalcBoundingBox(), PtSurfaceCalcBoundingBoxById()	Вычисляет охватывающий прямоугольник для поверхности управления
PtSurfaceExtent(), PtSurfaceExtentById()	Вычисляет пределы поверхности управления
PtSurfaceHit()	Находит поверхность управления для заданной точки
PtSurfaceRect(), PtSurfaceRectById()	Получает ограничивающий прямоугольник для поверхности управления

PtSurfaceTestPoint() Проверяет, находится ли точка на поверхности управления

Прорисовка поверхностей управления

Поверхности управления опрашиваются на самопрорисовку от заднего плана вперёд, после того как виджет перерисовал себя. Никакого отсечения для Вас не производится. Если Вы хотите отсечения, Вы осуществляете необходимую логику подгонки списка отсечений по ходу прохождения поверхностей, и затем восстанавливаете стек отсечений после того, как прорисована последняя поверхность. В противном случае Вы получите какие-то непредсказуемые результаты.

Следующие функции инициируют перерисовку поверхностей управления:

PtDamageSurface(), Помечает поверхность как повреждённую, так что она
PtDamageSurfaceById() будет перерисована.
PtDamageSurfaceByAction() Повреждает все поверхности, связанные с действием.

Активация поверхностей управления

Эта функция активирует поверхность управления:

PtCheckSurfaces() Сопоставляет событие с поверхностями управления, принадлежащими виджету.

Включение и отключение поверхностей управления

Вы можете включать и отключать поверхности управления, подобно виджетам:

PtDisableSurface(), PtDisableSurfaceById() Отключает поверхность управления
PtDisableSurfaceByAction() Отключает все поверхности управления, связанные с действием
PtEnableSurface(), PtEnableSurfaceById() Включает поверхность управления
PtEnableSurfaceByAction() Включает все поверхности управления, связанные с действием
PtSurfaceIsDisabled() Определяет, отключена ли поверхность управления
PtSurfaceIsEnabled() Определяет, включена ли поверхность управления

Нахождение поверхностей управления

Чтобы найти поверхность управления, используйте эти функции:

PtFindSurface() Отыскивает поверхность управления с заданным идентификатором
PtFindSurfaceByAction() Ищет поверхность управления, связанную с заданным действием
PtWidgetActiveSurface() Получает текущую активную в данный момент поверхность управления виджета

Скрытие и демонстрация поверхностей управления

Вы можете также скрыть и показать поверхности управления:

PtHideSurface(), PtHideSurfaceById() Скрывает поверхность управления
PtHideSurfaceByAction() Скрывает все поверхности управления, связанные с действием
PtShowSurface(), PtShowSurfaceById() Показывает скрытую поверхность управления
PtShowSurfaceByAction() Показывает все скрытые поверхности управления, связанные с акцией
PtSurfaceIsHidden() Определяет, скрытой ли является поверхность управления
PtSurfaceIsShown() Определяет, видна ли поверхность управления

Установка порядка поверхностей управления

Как и в случае виджетов, Вы можете собирать поверхности управления в стек:

PtInsertSurface(),	Вставляет поверхность управления впереди или позади другой
PtInsertSurfaceById()	
PtSurfaceBrotherBehind()	Получает поверхность управления, находящуюся позади данной
PtSurfaceBrotherInFront()	Получает поверхность управления, находящуюся впереди заданной
PtSurfaceInBack()	Получает самую заднюю поверхность управления, принадлежащую виджету
PtSurfaceInFront()	Получает самую переднюю поверхность управления, принадлежащую виджету
PtSurfaceToBack(),	Перемещает поверхность управления назад за все другие поверхности
PtSurfaceToBackById()	управления, принадлежащие виджету
PtSurfaceToFront(),	Перемещает поверхность управления впереди всех других
PtSurfaceToFrontById()	поверхностей управления, принадлежащих виджету

Размещение пользовательских данных вместе с поверхностями управления

В функциях, связанных с поверхностями управления, нет данных по ответным реакциям; Вы можете размещать данные пользователя вместе с поверхностями управления, вызывая:

PtSurfaceAddData(),	Добавляет данные к поверхности управления
PtSurfaceAddDataById()	
PtSurfaceGetData(),	Получает данные, связанные с поверхностью управления
PtSurfaceGetDataById()	
PtSurfaceRemoveData(),	Удаляет данные из поверхности управления
PtSurfaceRemoveDataById()	

Пример

Вот Вам программа, создающая несколько поверхностей управления:

```
#include <Pt.h>

/* Это функция, вызываемая, когда для нашей прямоугольной поверхности
управления происходит событие. Когда пользователь щёлкает
на этой поверхности, мы подсчитаем итог и напечатаем, сколько раз это приключилось. */

static int rect_surface_callback( PtWidget_t *widget, PtSurface_t *surface, PhEvent_t *event) {
static int rclicks = 1;
printf("Щелчков по прямоугольнику: %d\n", rclicks++);
return(Pt_END);
}

/* Эта функция, рисующая содержание нашей прямоугольной поверхности
управления. Это очень простой пример; он рисует красный прямоугольник. */

static void rect_surface_draw( PtWidget_t *widget, PtSurface_t *surface, PhTile_t *damage) {
PgSetFillColor(Pg_RED);
PgDrawRect(PtSurfaceRect(surface, NULL), Pg_DRAW_FILL);
}

/* Это функция, синхронизирующая размер поверхности управления с размером виджета.
PtWidgetExtent() возвращает прямоугольник, охватывающий текущий размер виджета.

PtSurfaceRect() является макросом; это означает, что Вы имеете прямой доступ к данным
внутри Вашей поверхности управления. Вам нет нужды вызывать какую-либо функцию,
чтобы изменить её размер. Изменяйте напрямую данные. */

static void rect_surface_calc( PtWidget_t *widget, PtSurface_t *surface, uint8_t post) {
/* Делать это только после того, как виджет расширится. */
if (post) {
/* Прямоугольник занимает верхний левый квадрант окна. */
PhRect_t *extent;
PhRect_t *srect;

extent = PtWidgetExtent(widget, NULL);
srect = PtSurfaceRect(surface, NULL);
```

```

    srect->ul = extent->ul;
    srect->lr.x = (extent->ul.x + extent->lr.x) / 2;
    srect->lr.y = (extent->ul.y + extent->lr.y) / 2;
}}

/* Это функция, вызываемая, когда случается событие для нашей эллиптической
поверхности управления. Когда пользователь щёлкает на этой поверхности, мы
подсчитаем итог и напечатаем, сколько раз это приключилось. */

static int ell_surface_callback( PtWidget_t *widget, PtSurface_t *surface, PhEvent_t *event) {
    static int eclicks = 1;
    printf("Ellipse clicks: %d\n", eclicks++);
    return(Pt_END);
}

/* Эта функция, рисующая содержание нашей эллиптической поверхности
управления. Это очень простой пример; он рисует зелёный эллипс. */

static void ell_surface_draw( PtWidget_t *widget, PtSurface_t *surface, PhTile_t *damage) {
    PhRect_t *s = PtSurfaceRect(surface, NULL);
    PgSetFillColor(Pg_GREEN);
    PgDrawEllipse(&(s->ul), &(s->lr), Pg_DRAW_FILL | Pg_EXTENT_BASED);
}

/* Это наша main-функция. Мы создаём окно, инициализируем наше приложение
с сервером Photon и создаём две поверхности управления.

Заметьте, что вторая поверхность не обеспечивает последний параметр, функцию
вычисления занимаемого пространства. Это не нужно, так как пятый параметр,
высота и ширина хранятся в указываемой структуре. Это указатель на реальную
указываемую структуру внутри виджета окна. Поэтому, если объём окна изменился,
изменяя структуру указания пространства, поверхность управления обновляет свои
значения автоматически! */

int main(int argc, char **argv) {
    PtArg_t args[1];
    PtWidget_t *window;
    const PhDim_t dim = { 200, 200 };

    PtSetArg(&args[0], Pt_ARG_DIM, &dim, 0);
    window = PtAppInit(NULL, &argc, argv, 1, args);

    /* Создание прямоугольной поверхности управления. */
    PtCreateSurface( window, 0, 0, Pt_SURFACE_RECT, NULL, Ph_EV_BUTTON_PRESS,
                    rect_surface_callback, rect_surface_draw, rect_surface_calc);

    /* Создание эллиптической поверхности управления для заполнения окна. */
    PtCreateSurface( window, 0, 0, Pt_SURFACE_ELLIPSE,
                    (PhPoint_t*)PtWidgetExtent(window, NULL),
                    Ph_EV_BUTTON_PRESS, ell_surface_callback, ell_surface_draw, NULL);

    PtRealizeWidget(window);
    PtMainLoop();

    return(EXIT_SUCCESS);
}

```

Глава 13. Доступ к модулям PhAB из программного кода

В этой главе обсуждается:

- Создание внутренних связей
- Использование внутренних связей в Вашем коде
- Базы данных виджетов

Вы можете получить доступ к любому модулю напрямую из программного кода Вашего приложения, создав внутреннюю связь к этому модулю.

Внутренняя связь подобна связи ответной реакции – она позволяет Вам задать тип модуля, функцию предустановки и там, где это присуще, расположение. Но в отличие от связи ответной реакции, которая всегда привязана непосредственно к ответной реакции виджета, внутренняя связь не привязана ни к какому виджету. Вместо этого PhAB сгенерирует описание, которое Вы используете в своём программном коде, чтобы задать, какую внутреннюю связь Вы хотите использовать. PhAB предоставляет несколько функций, помогающих Вам использовать внутренние связи (обсуждаются ниже).

Вы можете использовать внутренние связи, чтобы:

- Создать модуль PhAB внутри кода приложения. Используя связь ответной реакции, Вы можете напрямую связать виджет с модулем приложения PhAB. Но иногда Вам необходимо вместо этого создавать модуль из Вашего программного кода. Чтобы это сделать, используйте внутреннюю связь. Вот несколько общих ситуаций, когда Вы будете использовать внутреннюю связь для создания модуля:
 - Когда Ваше приложение может отображать один из двух различных модулей, основанных на одном и том же состоянии внутри приложения.
 - Когда Вам надо управлять генеалогией модуля, вместо того чтобы использовать принимаемую в PhAB по умолчанию (по умолчанию новый модуль является потомком базового окна).
 - Когда Вы хотите отобразить меню при нажатии пользователем правой кнопки мыши.
- Получить доступ и отобразить модули картинок. Вы используете модули картинок главным образом для того, чтобы заменить содержимое существующих контейнерных виджетов, таких как PtWindow или PtPanelGroup. Заметьте, что когда Вы создаёте модуль картинки, используя ApCreateModule(), Вы должны задать родительский контейнерный виджет.
- Открыть базы данных виджетов. Более подробно см. "Использование баз данных виджетов".

Создание внутренних связей

Чтобы создать внутреннюю связь:

1. Выберите пункт "Internal Links" в меню "Application" или нажмите <F4>. Вы увидите диалог "Internal Module Links":

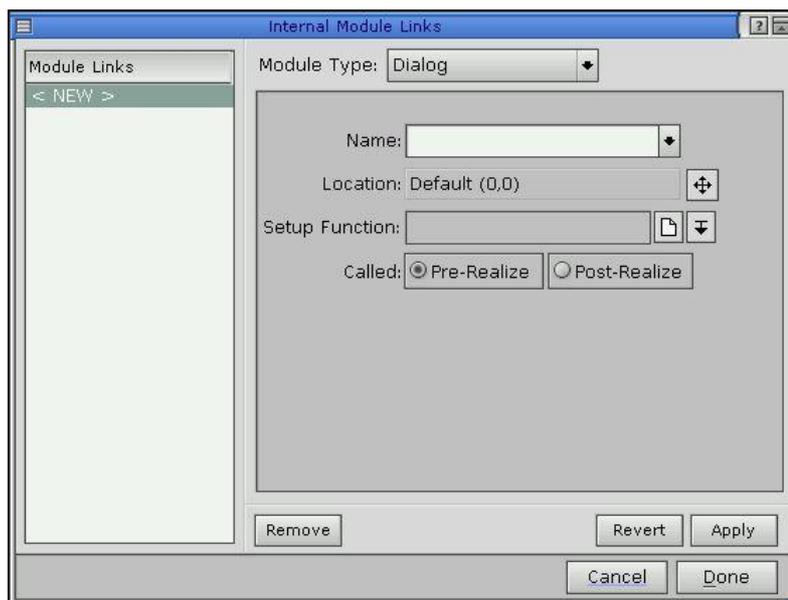


Рис. 13-1. Диалог "Internal Module Links"

2. Щёлкните на опции <NEW>, если она ещё не выбрана
3. Выберите требующийся Вам тип модуля
4. Заполните области в разделе информации о связи модуля – см. ниже
5. Щёлкните на "Apply", затем щёлкните на "Done"

☞ Вы можете создавать только по одной внутренней связи для каждого модуля.

Области диалога "Internal Module Links" включают:

- Name – Содержит имя модуля. Чтобы выбрать имя из списка существующих модулей, щёлкните на кнопке рядом с этой областью.
- Location – определяет, где появится этот модуль; см. "Позиционирование модуля" в главе "Работа с модулями".
- Setup Functaiion – задаёт функцию, которая будет вызываться, когда модуль реализуется (необязательная). Чтобы редактировать функцию, щёлкните на иконке рядом с этой областью. Более подробно см. в разделе "Установочные функции модуля" в главе "Работа с программным кодом".
- Called – Определяет, вызывается ли установочная функция перед тем, как модуль реализуется, после того, или – и до и после.
- Apply – Принимает все изменения
- Reset – Восстанавливает всю информацию о внутренней связи в её первоначальном состоянии.
- Remove – Удаляет выбранную внутреннюю связь из списка связей модулей.

Использование внутренних связей в Вашем программном коде

Декларации

Для каждой внутренней связи, определённой в Вашем приложении, PhAB генерирует декларацию, так чтобы Вы могли идентифицировать и получить доступ к связи.

Поскольку PhAB извлекает имя декларации из имени модуля, каждый модуль может иметь только одну внутреннюю связь. Это может показаться ограничением, но PhAB предоставляет относящиеся к модулям функции (см. ниже), позволяющие Вам изнутри Вашего программного кода переделать установочную функцию модуля и месторасположение.

При создании имени декларации PhAB берёт имя модуля и добавляет ABM_ в качестве префикса. Так, например, если Вы создаёте внутреннюю связь к модулю с именем mydialog, PhAB создаст декларацию ABM_mydialog.

Функции внутренней связи

Декларация используется следующими функциями API PhAB'a:

ApCreateModule()	Позволяет Вам вручную создавать модули, спроектированные в PhAB'e. Модуль, созданный этой функцией, ведёт себя точно так же, как если бы он был непосредственно связан связью ответной реакции. Например, если Вы определяете месторасположение и установочную функцию для внутренней связи, модуль появится в этом заданном месте и будет вызвана установочная функция. Более того, ответные реакции виджетов, горячие клавиши и всё такое прочее тоже активизируется.
ApModuleFunction()	Позволяет Вам изменить установочную функцию, присоединённую к внутренней связи.
ApModuleLocation()	Позволяет Вам изменить местоположение модуля на экране, присоединённое к внутренней связи.
ApModuleParent()	Позволяет Вам изменить родителя модуля окна или диалога, связанного внутренней связью. Эта функция употребляется только для внутренних связей к модулям окон и диалогов.
ApOpenDBase()	Позволяет Вам открыть модуль, присоединённый через внутреннюю связь как база данных виджетов.

Для получения более полной информации об этих функциях см. "Справочник библиотечных функций Photon'a".

Пример – отображение меню

Вот так Вы можете отобразить меню, когда пользователь нажмёт правую кнопку мыши, поместив её указатель на виджет:

1. В PhAB создайте модуль меню. Дайте ему имя, скажем my_menu.
2. Создайте внутреннюю связь (см. раздел "Создание внутренних связей" настоящей главы), как описано выше. Для выпадающего меню, обычно Вам потребуется, чтобы модуль располагался относительно виджета либо относительно указателя мыши.
3. Выберите виджет, связанный с меню. Убедитесь, что в его Pt_ARG_FLAGS установлен флаг Pt_MENUABLE и сброшен Pt_ALL_BUTTONS.
4. Сгенерируйте код для Вашего приложения. PhAB создаст декларацию внутренней связи. В нашем случае она будет называться ABM_my_menu.
5. Каждый виджет, являющийся потомком PtBasic, имеет ресурс Pt_CB_MENU, который представляет из себя список ответных реакций, вызываемых, когда Вы нажимаете правую клавишу мыши при указателе мыши на виджете. Отредактируйте этот ресурс и создайте функцию ответной реакции, подобную этой:

```
int text_menu_cb( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo ) {
/* предотвращает предупреждения (варнинги) об отсутствии ссылок */
widget = widget, apinfo = apinfo, cbinfo = cbinfo;

ApCreateModule (ABM_my_menu, widget, cbinfo);

return( Pt_CONTINUE );
}
```

Параметр *widget*, передаваемый функции `ApCreateModule()`, используется, если меню позиционируется относительно виджета; аргумент *cbinfo* – если относительно указателя мыши.

6. Скомпилируйте, слинкуйте и запустите Ваше приложение. Когда Вы нажмёте правую клавишу на виджете, появится Ваше меню.

Использование базы данных виджетов

Модули картинок служат двум целям:

- позволять приложению заменять содержание какого-либо контейнерного виджета;
- служить в качестве баз данных виджетов.

Если Вы планируете использовать виджет в Вашем приложении несколько раз, база данных виджетов позволит Вам спроектировать виджет только однажды. Она также убергает Вас от необходимости писать кучу кода. Всё, что Вы делаете – это предустанавливаете ресурсы виджета, и затем, используя функции API базы данных виджетов PhAB'a, создаёте копию виджета всякий раз, когда обычно создаёте виджет из своего программного кода.

Вот пример базы данных виджетов – это является частью базы данных виджетов, которую PhAB использует в своём собственном интерфейсе:

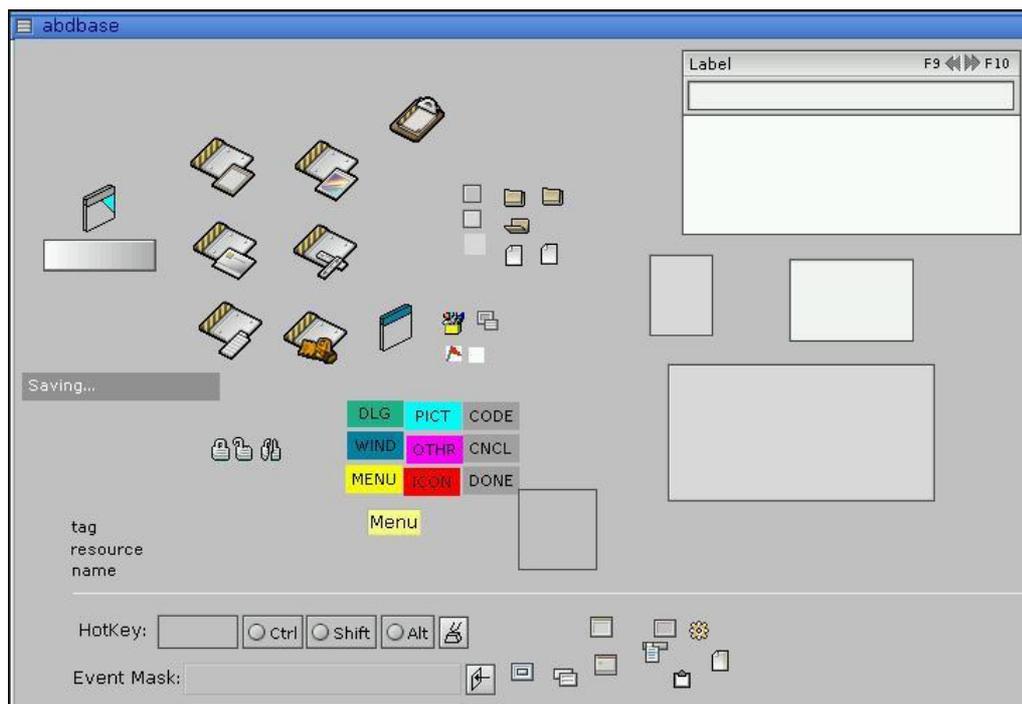


Рис. 13-2. Пример базы данных виджетов

Создание базы данных

Чтобы создать базу данных виджетов:

1. Создайте модуль картинки в своём приложении
2. Создайте внутреннюю связь к модулю картинки
3. Создайте виджеты, к которым Вам необходимо иметь доступ из Вашего программного кода.

Например, скажем, Вам надо в Вашем приложении много раз создавать некую иконку. Создав иконку внутри модуля картинки, Вы сможете создать при выполнении программы столько копий иконки, сколько Вам понадобится.

Предварительно прикрепленные ответные реакции

Кроме того, что можно предустановить все ресурсы виджета в модуле базы данных, Вы также можете предварительно прикрепить их ответные реакции. Когда Вы создаёте виджет динамически, будут также созданы все прикрепленные Вами ответные реакции.

Предустанавливая ресурсы и ответные реакции для виджета базы данных, Вы легко можете уменьшить объём кода, требуемого для динамического создания виджета, до одной-единственной строки.

☞ Предварительное прикрепление ответных реакций работает только с модулями и функциями, которые являются частью Вашего исполняемого файла. Если Ваше приложение открывает внешний файл как базу данных виджетов, библиотека PhAB'a не будет в состоянии найти код, чтобы прикрепить ответную реакцию.

Назначение уникального имени экземпляра

Присвойте каждому виджету в базе данных виджетов уникальное имя – это позволит Вам ссылаться на виджеты при использовании функций API, относящихся к базам данным.

Создание динамической базы данных

Вы также можете создать базу данных виджетов, которую Вы сможете изменять динамически. Чтобы сделать это, откройте внешнюю базу данных виджетов – т.е. такую, которая не подвязана к Вашему исполняемому файлу – функцией `ApOpenDBaseFile()` вместо вызова функции `ApOpenDBase()`. Функция `ApOpenDBaseFile()` позволяет Вам получить непосредственный доступ к файлу модуля и открыть его как базу данных.

Открыв однажды файл модуля, Вы сможете копировать виджеты из этого файла в Вашу внутреннюю базу данных приложения и сохранять результирующую базу данных в новом файле, который Вы сможете впоследствии переоткрыть.

Функции базы данных виджетов

PhAB предлагает несколько вспомогательных функций, позволяющих Вам открыть базу данных виджетов и скопировать её виджеты в модули – Вы можете скопировать виджеты столько раз, сколько Вам понадобится. PhAB также предоставляет удобные функции, позволяющие Вам копировать виджеты между базами данных, создавать виджеты, удалять их, и сохранять базы данных виджетов.

<code>ApOpenDBase(),</code> <code>ApCloseDBase()</code>	Они позволяют Вам открывать и закрывать базу данных виджетов. Чтобы сразу обеспечить доступность базы данных, обычно используется функция <code>ApOpenDBase()</code> в функции инициализации приложения.
--	--

<code>ApOpenDBaseFile(),</code> <code>ApSaveDBaseFile()</code>	Они позволяют Вам открыть и сохранить файл внешнего модуля как базу данных внутри Вашего приложения.
---	--

ApAddClass()	Эта функция позволяет Вам указать, с какими классами виджетов Вы, вероятно, столкнётесь, когда вызовете ApOpenDBaseFile(). При линковке Вашего приложения только те виджеты, которые Вам требуются, будут подлинкованы к Вашему приложению. Если Вы получаете доступ к виджетам, которых нет в Вашем приложении, поскольку они находятся во внешней базе данных, Вы должны добавить их ко внутренней таблице классов, так чтобы они подлинковались на этапе компилирования.
ApCreateDBWidget(), ApCreateDBWidgetFamily(), ApCreateWidget(), ApCreateWidgetFamily()	Они создают виджеты из базы данных виджетов. Функции ApCreateWidget() и ApCreateDBWidget() создают только один виджет, невзирая на класс виджета. Для виджета неконтейнерного класса функции ApCreateWidgetFamily() и ApCreateDBWidgetFamily() создают одиночный виджет; для виджета контейнерного класса они создают все виджеты внутри контейнера. Эти функции отличаются по родителю, используемому для виджетов: <ul style="list-style-type: none">• Функции ApCreateDBWidget() и ApCreateDBWidgetFamily() используют аргумент parent; если он установлен в NULL, виджет не имеет родителя.• Функции CreateWidget() и ApCreateWidgetFamily() отдают новый виджет(ы) текущему родителю. Чтобы быть уверенным в правильности передачи виджета текущему родителю, перед вызовом любой из этих двух функций выполните вызов функции PtSetParentWidget(). <p>☞ Не используйте декларации, генерируемые для модуля картинки базы данных виджетов. Вместо этого используйте указатели, возвращаемые функцией ApCreateWidget() или ApCreateDBWidget().</p>
ApCopyDBWidget()	Позволяет Вам копировать виджет из одной базы данных виджетов в другую. Обычно Вы используете эту функцию только тогда, когда динамически создаёте или сохраняете базу данных виджетов внутри Вашего приложения.
ApDeleteDBWidget()	Удаляет виджет из базы данных виджетов
ApGetDBWidgetInfo()	Получает информацию о виджете в базе данных виджетов, включая его имя, класс, родителя и уровень иерархии.
ApGetImageRes()	Извлекает данные по ресурсам образа из виджета и использует эти данные для установки виджета, уже изображённого в Вашем приложении. Эта функция позволяет Вам добиться простейшей анимации. <p>☞ Если вы используете базу данных виджетов для создания виджетов, имеющих данные PhImage_t, прикрепленные к ним, не закрывайте базу данных функцией ApCloseDBase() до тех пор, пока эти виджеты не будут уничтожены. (Закрытие базы данных освобождает память, используемую для образа). Если Вы должны закрыть базу данных, убедитесь, что скопировали данные образа внутри программного кода Вашего приложения, и переустановите ресурс данных образа, так чтобы он указывал на Вашу новую копию.</p>

ApGetTextRes()

Эта функция позволяет Вам извлекать текстовую строку из базы данных виджета. Это полезно для многоязычных приложений, когда текст автоматически переводится, если включена поддержка языка. Для получения более полной информации см. приложение "Поддержка международных языков".

ApRemoveClass()

Удаляет класс виджета. Если Вы загрузили DLL, которая определяет класс виджетов, Вы должны будете удалить их перед выгрузкой DLL. Для более полной информации см. раздел "Существование DLL в приложениях PhAB" главы "Генерация, компиляция и запуск программного кода на выполнение".

Для получения более полной информации о функциях баз данных виджетов см. "Справочник библиотечных функций Photon".

Глава 14. Поддержка международных языков

PhAB имеет встроенную поддержку для приложений, требующих перевода на другие языки. Эта глава включает:

- Соображения о проектировании приложений
- Генерация языковой базы данных
- Базы данных сообщений
- Редактор языков
- Запуск Вашего приложения на исполнение
- Распространение Вашего приложения

Помня о нескольких соображениях в отношении проектирования, и затем следуя нескольким простым шагам, Вы сможете очень легко перевести Ваше приложение на другой язык без необходимости перекомпиляции или перестройки Вашего приложения:

1. PhAB генерирует базу данных для всех текстовых строк, встречающихся в Вашем приложении.
2. Эта текстовая база данных используется редактором языков PhAB'a, чтобы дать Вам возможность переводить каждую текстовую строку на другой язык.
3. Переведенные текстовые строки сохраняются в файле перевода и добавляются в Ваше приложение.
4. Чтобы запустить Ваше приложение на исполнение с другим языком, просто установите переменную окружения перед запуском приложения. Когда API PhAB'a построит окна приложения, диалоги и другие модули, он заменит текстовые строки другими, с новыми переводами. [Прим.пер.: не совсем так. Переменная окружения ABLANG "принимается" на этапе запуска сессии Photon'a, и экспорт нового значения влияния уже не оказывает. Поэтому не экспортом переменной, а записью соответствующей строки в файл /root/.ph/.ABLANG, из которого берёт значение переменной строка из файла /root/.profile, выполняется переключение языка. После чего необходим перезапуск сессии Photon'a].

Это так просто.

Соображения о проектировании приложения

В этом разделе представлены несколько соображений по поводу проектирования, которые помогут Вам создать приложение, независимое от языка. Проектируя и реализовывая Ваше приложение, Вы должны иметь в виду эти соображения, поскольку модификация приложения после её завершения будет более трудной.

Размер виджетов, основанных на тексте

Обычно, когда Вы проектируете приложение, Вы планируете окно, использующее виджеты, имеющие уже предустановленный по умолчанию текст приложения. Например, если Вы имеете кнопку "Done" в нижней части окна диалога, сама кнопка должна быть достаточно большой лишь в той мере, чтобы разместить текстовую строку "Done". Вы также разместите кнопку "Done" на основании её текущего размера. Это хорошо работает в приложениях, которые не требуют перевода, но станет причиной многих проблем в приложении, независимом от языка. Что будет, если переведенный текст будет состоять не из 4 символов, как по умолчанию, а из 12?

- "Переводимая" кнопка может стать больше по размеру. В этом случае она может стать такой широкой, что будет писать поверх других виджетов окна. Это приведёт к тому, что приложение будет выглядеть хреново или убого спроектированным

или

- Текст может быть обрезан внутри размера, заданного по умолчанию. В этом случае переводимый текст может стать нечитабельным, и пользователь не будет знать, что эта кнопка делает.

Например, эти кнопки слишком малы, чтобы быть приспособленными под переведенный текст:



Решение элементарно. Сделайте кнопки больше, чтобы подогнать их под более длинные переведенные строки. Вот так, например:

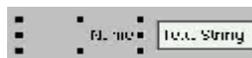


Выравнивание

Кроме создания основанных на тексте виджетах достаточной ширины для приспособления к переведенному тексту, Вы должны подумать о выравнивании текста, опираясь на использование его в виджете. Например, в области ввода одиночной текстовой строки, достаточно общим является размещение обозначающей надписи слева от области. Если вы делаете эту надпись широкой, чтобы обеспечить переводную надпись, сама надпись перемещается далеко влево:



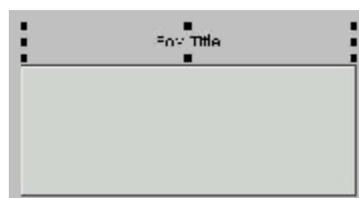
Эта проблема легко решается установкой горизонтального выравнивания надписи в выравнивание справа. Это допускает более длинные переведенные текстовые строки и по-прежнему удерживает плотное выравнивание с областью ввода текста:



Другим общеприменимым методом размещения надписей является центрирование надписи сверху или внутри границ охватывающего прямоугольника. Обычно текст центрируется путём размещения его в желаемой позиции, основываясь на его текущем тексте:



Когда впоследствии текст переводится, он оказывается либо короче, либо длиннее, и надпись над прямоугольником выглядит скособоченной. Простым решением является сделать обрамляющий прямоугольник надписи большим, чем это необходимо, и установить центрирование по горизонтальному выравниванию.



Вероятно, существует множество других случаев, подобных этому, но важным этапом разработки является необходимость обдумать, как переведенный текст скажется на виде приложения. По большей части эстетика может быть выдержана простым созданием виджетов, основанных на тексте, достаточно широкими по размеру и установкой для них соответствующего выравнивания.

Высота шрифта

Шрифты некоторых языков, таких как японский или китайский [Чур меня, чур – Прим. пер.], читабельны только при большом размере шрифта. Для таких шрифтов минимальным размером может быть 14 пунктов или даже больше. Если Вы спроектировали всё Ваше приложение, используя шрифт Helvetica в 10 пунктов, у Вас появится куча проблем, когда все Ваши базирующиеся на тексте виджеты растянутся на 4 пикселя и больше, чтобы подстроится под большие по размеру шрифты. Если Ваше приложение должно переводиться на другие языки, посмотрите на требования к шрифтам, прежде чем начнёте работать, и используйте минимальный размер шрифта принятого по умолчанию языка, встроенного в приложение. Если Вы на самом деле хотите использовать для Вашего принимаемого по умолчанию в приложении текста меньшие размеры шрифтов, Вы можете позаимствовать рекомендации из раздела выше. Вы можете сделать высоту виджета больше и установить центрирование по вертикальному выравниванию. Однако это может не сработать для областей ввода текста, и Вы должны иметь это в виду.

Жёстко закодированные строки

Другой большой сферой обсуждения являются информационные, предупреждающие, сообщающие об ошибках и иные текстовые сообщения, отображающиеся во всплывающих диалоговых окнах или других местах приложения. Например, включим вызовы функций PtAlert(), PtNotice() и PtPrompt(). Наиболее общим способом обработки текстовых сообщений является вставка текстовых строк в исходный код приложения.

Например:

```
char *btns[] = { "&Yes", "&No", "&Cancel" };
answer = PtAlert( base_wgt, NULL, NULL, NULL, "File has changed. Save it?", NULL,
                 3, btns, NULL, 1, 3, Pt_MODAL );
```

Хотя это и быстро в коде, но невозможно перевести без переписывания исходного кода приложения, перекомпиляции и т.д. В сущности, Вам надо создать совершенно новую версию приложения под каждый поддерживаемый язык. Значительно лучшим методом является использование достоинств баз данных виджетов PhAB. Используя базу данных виджетов, Вы можете поместить все Ваши текстовые сообщения в одну (или несколько) базу данных и дать каждому сообщению уникальное имя. Чтобы извлечь текст во время исполнения, вызовите функцию ApGetTextRes() (подробности см. в "Справочнике библиотечных функций Photon") [– тем самым получив текстовый ресурс заданного виджета, расположенного в заданной базе данных виджетов. Прим.пер.]. В вышеприведенном случае это может быть:

```
char *btns[3];

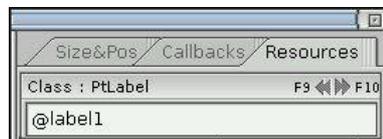
btns[0] = ApGetTextRes( textdb, "@msgyes" );
btns[1] = ApGetTextRes( textdb, "@msgno" );
btns[2] = ApGetTextRes( textdb, "@msgcancel" );
answer = PtAlert( base_wgt, NULL, NULL, NULL, ApGetTextRes( textdb, "@msg001" ), NULL, 3,
                 btns, NULL, 1, 3, Pt_MODAL );
```

Этот метод позволяет приложению не иметь предопределённых текстовых сообщений внутри него, и приложение может быть легко переведено. Кроме того, поскольку текстовые строки помещаются в базу данных виджетов, PhAB автоматически позаботится о включении текстовых сообщений, когда будет генерировать базу данных текстовых строк приложения. Это более удобно, чем простое использование внешнего текстового файла и проектирование какого-то другого способа перевода этого файла.

Использование символа "@" в именах экземпляров

По умолчанию PhAB игнорирует виджеты, которые не имеют имени экземпляра или имеют имя экземпляра, установленное в имя класса. Это означает, что когда Вы размещаете надпись внутри окна и изменяете текст на что-то соответствующее, PhAB пропускает этот виджет, когда генерирует исходный код Вашего приложения. Это происходит потому, что PhAB предполагает, что надпись является константой и приложению не понадобится получить к ней доступ. Однако, когда дело доходит до необходимости перевести Ваше приложение на другой язык, эта надпись становится очень важной.

Чтобы различать виджеты, которые важны в смысле перевода, но не в смысле генерации исходного кода, PhAB опознаёт специальный символ, когда тот расположен в первой позиции имени экземпляра. Этот специальный символ – "@". Это означает, что Вы можете присвоить надписи имя экземпляра "@label1", и PhAB опознает эту надпись при генерации базы данных разноязычных текстов, но пропустит при генерации исходного кода.



Это всё звучит хорошо, за исключением того, что PhAB также требует, чтобы все имена экземпляров были уникальными. Это правило должно выдерживаться, так чтобы PhAB при исполнении программы знал, какую текстовую строку заменять. К несчастью, выдумывание потенциально сотен уникальных имён экземпляров, которые в действительности Вас не интересуют, может оказаться кучей работы. Для решения этой задачи PhAB позволяет Вам задавать одиночный символ "@" в качестве имени экземпляра, и PhAB сам добавит сзади внутренний последовательный номер. Это освободит Вас от необходимости отслеживать все константные текстовые строки, которые требуют имён экземпляров только для перевода. Если Вы хотите сгруппировать переводные текстовые строки (скажем, в модуле), Вы можете дать им все одно имя экземпляра, и PhAB добавит последовательный номер, чтобы создать уникальное имя. Например, если Вы назначили имя "@base" нескольким виджетам, PhAB сгенерирует в качестве имён экземпляров "@base", "@base0", "@base1" и т.д.

Двужычные приложения

Иногда необходимо спроектировать двужычное приложение. Это предполагает, что в каждой текстовой строке отображается два различных языка. Хотя это и можно сделать, обычно пользователю тяжело такое читать и понимать.

PhAB предлагает Вам использовать другой подход. Вы можете создать приложение в одном языке и обеспечить возможность переключения на другой через управление приложением. Это выполняется через функцию API PhAB – функцию `ApSetTranslation()`. Эта функция (описание см. в "Справочнике библиотечных функций Photon'a") изменяет текущий файл перевода для приложения немедленно, так что все *последующие* диалоги, окна и всё такое прочее рисуются с использованием нового файла перевода.

☞ Любые уже существующие модули и виджеты при этом не переводятся, только новые. Если Вы хотите добиться немедленной обратной связи, Вам понадобится пересоздать модули. Это просто для диалогов, но сложнее с базовым окном; вспомните, что уничтожение базового окна приводит к завершению приложения. Единственным способом перевести содержание базового окна – это поместить всё содержание в модуль картинки, который может быть пересоздан.

Общие строки

Если у Вас есть несколько приложений для перевода, Вы можете уменьшить объём работы, совместно используя общие текстовые строки и переводя их отдельно. Чтобы это сделать:

1. Создайте самостоятельное приложение, которое содержит один модуль картинки (базу данных виджетов) общих текстовых строк.
2. Используйте редактор языка PhAB для перевода строк.
3. Как только база данных создана и переведена, получите доступ к ней из другого приложения, вызвав функцию `ArAppendTranslation()` (описанный в "Справочнике библиотечных функций Photon'a").

Генерация языковой базы данных

Это лёгкая часть работы. Наиболее важным аспектом на этом шаге – это знать, когда генерировать базу данных текстовых строк. В идеале сделать это надо тогда, когда завершена вся разработка приложения, поскольку механизм перевода при исполнении приложения вращается вокруг имени экземпляра виджета. Если Вы генерируете Вашу базу данных в середине процесса разработки и выполняете перевод, вполне вероятно, что куча виджетов изменится или будет удалена, и переводы придётся удалить или время будет потеряно.

Единственным исключением из этого являются двуязычные приложения. В этом случае Вы можете генерировать и переводить приложение непрерывно, так что перевод приложения может быть оттестирован на протяжении разработки.

Чтобы сгенерировать языковую базу данных приложения:

1. Выберите меню "Application"
2. Выберите пункт "Languages/Generate Language database". Всплывёт прогрессирующий диалог, и немедленно будет сгенерирована база данных.
3. Щёлкните на "Done", когда генерация завершится.

☞ Пункт "Languages" в меню "Application" недоступен, если Вы не сохранили Ваше приложение в первый раз, присвоив ему имя.

Теперь база данных сгенерирована и готова для использования языковым редактором PhAB. Имя базы данных – "app.ldb", где "app" – это имя исполняемого файла приложения (которое обычно такое же, как и имя приложения, если Вы не использовали команду "Save As", чтобы переименовать приложение [В противном случае это имя оригинала – Прим.пер.]). Языковая база данных помещается в директорию приложения (туда, где найден файл `abapp.dfn`).

Базы данных сообщений

База данных сообщений – это файл, содержащий текстовые строки. Каждое сообщение идентифицируется именем тэга. Чтобы загрузить базу данных сообщений, вызывается функция `ArLoadMessageDB()`. Эта функция выполняет обычный поиск файла, основываясь на переменной окружения `ABL_PATH` и текущем языке:

- Если никакой язык не определён, база данных сообщений загружена. Она должна иметь имя "name.mdb".
- Если язык определён, функция просматривает файл перевода с именем "name.language". Файлы перевода могут быть созданы, используя редактор-переводчик PhAB'a – он может обрабатывать базы данных сообщений.

Чтобы получить сообщение по его тэгу, используется функция `ApGetMessage()`.

Чтобы закрыть базу данных сообщений, используется вызов функции `ApCloseMessageDB()`.

Редактор языка

После того как база данных сгенерирована, Вы можете использовать языковой редактор PhAB'a, чтобы перевести принимаемую по умолчанию текстовую строку на другой язык. Языковой редактор спроектирован для работы и как автономное приложение, которое Вы можете распространять вместе с Вашим приложением, и как интегрированная часть собственно PhAB'a.



Рис. 14-1. Языковой редактор PhAB

Запуск языкового редактора из PhAB

Когда Вы разрабатываете приложение, находясь в среде PhAB, Вы можете совершенно элементарно запустить языковой редактор, использующий текущую базу данных языков приложения:

1. Выберите "Application Menu"
2. Выберите пункт "Select Languages/Run Language Editor".

Это запустит языковой редактор, использующий текущую базу данных языков приложения. В этом месте Вы можете выполнить создание нового файла перевода или редактировать имеющийся.

Запуск языкового редактора как автономного приложения

Если Вы планируете позволить выполнять перевод Вашего приложения со стороны клиента, Вам понадобится включить в Ваше приложение следующие файлы:

- /usr/photon/appbuilder/phablang
- /usr/photon/appbuilder/languages.def
- ☞ Файл `languages.def` должен находиться в той же директории, что и языковой редактор `phablang`.

- файл базы данных языков Вашего приложения, `xxx.ldb`

Чтобы запуститься со стороны клиента, Вы можете:

- набрать команду `/usr/photon/appbuilder/phablang &`
или
- создать вход в менеджере рабочего стола (DesktopManager) для запуска `/usr/photon/appbuilder/phablang` (предполагая, что клиент запускается в полном окружении рабочего стола).

После запуска phablant:

1. Щёлкните на иконке "Open Folder" чтобы вызвать файловый селектор.



2. Используя файловый селектор, найдите файл приложения xxx.ldb.
3. Откройте файл xxx.ldb.

Создание нового файла перевода

Чтобы создать новый файл перевода:

1. Щёлкните на кнопке "New", расположенной в нижней части окна. Отобразится диалог "Language Selection".



Рис. 14-2. Диалог "Language Selection"

2. Выберите желаемый язык из списка поддерживаемых типов языковых файлов.
3. Щёлкните на "Add".
4. В этом месте Вам потребуется подтвердить свой выбор. Щёлкните на "Yes".

Диалог "Language Selection" закроется, и Вы теперь увидите вновь созданный файл перевода в списке доступных переводов.

Редактирование существующего файла перевода

Чтобы отредактировать файл перевода в списке переводов [список "Translations" диалога "PhAB Language Editor" – прим. пер.]

1. Щёлкните на нужном языке в списке.
2. Щёлкните на клавише "Open".

Появится диалог "Text Translation Editor". Этот текстовый редактор отображает все текстовые строки, доступные для перевода, в текущей базе данных языков.

Перевод текста

Чтобы перевести текстовую строку:

1. Щёлкните на текстовой строке, которую Вы хотите перевести. Выбранная текстовая строка отобразится в текстовой области в верхней части окна:



Default text принимаемый по умолчанию текст, привязанный к приложению

Translation текущий перевод (если имеется) для текстовой строки. Это область, используемая Вами для набора нового перевода.

- Если Вам необходимо набрать символ, которого нет на Вашей клавиатуре, Вы можете использовать составную последовательность, описанную в "Составные последовательности Photon'a" в Приложении "Поддержка многоязычности Unicode".
- Вам не надо переводить каждую строку. Если перевода нет, используется принимаемый по умолчанию текст.

При редактировании перевода Вы можете использовать кнопки вырезки, копирования и вставки, расположенные над областью перевода.

2. Сразу по изменении переводимой строки, выше области перевода появятся кнопки с зелёной птичкой и красным крестиком.
3. Щёлкните на зелёной птичке, чтобы принять Ваши изменения (кнопкой быстрого доступа является <Ctrl> – <Enter>).
4. Щёлкните на красном крестике, чтобы отменить Ваши изменения.

Повторите вышеприведенные шаги для всех строк, которые Вы хотите перевести. Когда закончите, щёлкните на кнопке "Save & Close".

Горячие клавиши

При переводе приложения появляется проблема, связанная с тем, что назначенные горячие клавиши больше не соответствуют переводимой строке, если та не включает значение клавиши быстрого доступа. Исходя из этого соображения, PhAB добавляет в базу данных языка также и строки клавиши быстрого доступа.

При переводе текстовой строки переводчик может также изменить клавишу быстрого доступа. Если клавиша, используемая в качестве горячей, не является функциональной (т.е. код клавиши меньше 0xF000), PhAB автоматически изменит горячую клавишу, чтобы соответствовать клавише быстрого доступа.

Например, предположим, Ваше приложение имеет кнопку с надписью "Cancel". Вы устанавливаете в C значение ресурса Pt_ARG_ACCEL_KEY, и монтируете <Alt> – <C> для вызова ответной реакции Pt_CB_HOTKEY.

Когда Вы генерируете базу данных языков, Вы обнаружите, что она включает надпись на кнопке и её клавишу быстрого доступа. Если Вы переводите приложение на французский, надпись на кнопке станет "Annuler", так что горячая клавиша <Atl> – <C> больше не подходит. Просто переведите Pt_ARG_ACCEL_KEY на A, и горячей клавишей автоматически станет <Alt> – <A>, когда Вы запустите приложение на французском.

- Вы должны убедиться, что отсутствуют повторяющиеся клавиши быстрого доступа. Если это, к несчастью, случится, будет признана только первая заданная клавиша.

Ресурсы help'a

Если Вы используете просмотрщик помощи (Photon Helpviewer) для организации помощи в Вашем приложении и планируется обеспечить для Вашего приложения многоязычные файлы помощи, переводчик также может перевести так называемые маршруты тем помощи, чтобы указать правильные позиции внутри соответствующих файлов помощи.

Функции перевода

Вы можете строить свой собственный редактор языков, если обнаружите, что предлагаемый по умолчанию не соответствует Вашим потребностям. Вы найдёте эти функции полезными.

AlClearTranslation()	Удаляет все переводы в базе данных языков или сообщений
AlCloseDBase()	Закрывает базу данных языков или сообщений
AlGetEntry()	Получает вход из базы данных языков или сообщений
AlGetSize()	Получает число записей в базе данных языков или сообщений
AlOpenDBase()	Загружает базу данных языков или сообщений
AlReadTranslation()	Читает файл перевода в базе данных
AlSaveTranslation()	Сохраняет переводы в базе данных языков или сообщений
AlSetEntry()	Устанавливает переведенную строку для входа базы данных

Вы можете использовать эти функции, чтобы создать свой собственный редактор языков, или преобразовать базу данных языков в файл другого формата (например, так Вы сможете переслать файл в не-Photon'овскую или не-QNX'овскую систему для перевода).

Запуск Вашего приложения на исполнение

После того как база данных языков полностью переведена, последним шагом является запуск приложения.

Когда Вы создаёте файлы перевода, они помещаются в той же директории, что и файл Вашего приложения `abapp.dfn`. Вы можете рассматривать это как рабочие версии файлов. Когда Вы запускаете Ваше приложение из PhAB'a, это является используемыми Вами версиями.

Когда Вы запускаете Ваше приложение вне PhAB, оно просматривает файлы перевода в следующем порядке:

1. В директориях, перечисленных в переменной окружения `ABLPATH`, если она задана. Этот список имеет вид: `dir:dir:dir:dir`.
В отличие от переменной окружения `PATH`, текущая директория должна указываться точкой, а не пробелом. Пробел указывает директорию, где располагается исполняемый файл.
2. В той же директории, где располагается исполняемый файл, если переменная окружения `ABLPATH` не определена.

Для того, чтобы API PhAB знал, какой файл перевода Вы хотите использовать, Вы должны установить значение переменной `ABLANG` в одно из следующих значений:

ЯЗЫК	ЗНАЧЕНИЕ
Бельгийский французский	fr_Be
Канадский английский	en_CA
Канадский французский	fr_CA
Датский	da_DK
Голландский	nl_NL
Французский	fr_FR
Немецкий	de_DE
Итальянский	it_IT
Японский	ja_JP
Норвежский	no_NO
Польский	pl_PL
Португальский	pt_PT
Словацкий	sk_SK
Испанский	es_ES
Шведский	se_SE
Швейцарский французский	fr_CH
Швейцарский немецкий	de_CH
Английский английский	en_GB
Американский английский	en_US

☞ Этот список приведен на момент написания этого документа, но с тех пор мог быть обновлён. Чтобы получить последнюю версию, см. файл `/usr/photon/appbuilder/languages.def` [*Хе-хе. В безнадёжных поисках ru_RU – Прим.пер.*]

Например, чтобы запустить приложение на немецком (как выражаются, в Германии), Вы должны сделать следующее:

```
$ export ABLANG=de_DE
$ myapplication
```

☞ Приложение ищет наилучшую возможность совпадения. Например, если расширение языка задано как `fr_CA`, поиск выполняется следующим образом:

1. Точное совпадение (например, `fr_CA`);
2. Совпадение в основном (например, `fr`);
3. Совпадение по групповому символу (например, `fr*`).

Если никакого перевода не найдено, используется оригинальный текст приложения.

Команда `export` может быть помещена в профайл входящего пользователя, так что приложение будет запускаться на языке, предопределённом каждым пользователем.

Распространение Вашего приложения

Когда Вы отправляете Ваше приложение клиенту, Вы должны убедиться, что включили файлы переводов в Ваш лист отправки. Например, если Ваше приложение называется `myapp`, то у Вас есть файлы переводов на французский и немецкий, Вам надо включить в приложение файлы `myapp.fr_FR` и `myapp.de_DE`.

Эти файлы должны располагаться:

- В директориях, перечисленных в переменной окружения `ABLPATH`, если она задана. Этот список имеет вид:

```
dir:dir:dir:dir
```

В отличие от переменной окружения `PATH`, текущая директория должна указываться точкой, а не пробелом. Пробел указывает директорию, где располагается исполняемый файл.

- В той же директории, где располагается исполняемый файл, если переменная окружения `ABLPATH` не определена.

Если Вы хотите, чтобы каждый клиент мог переводить приложение, Вы также должны распространять с приложением:

- языковой редактор (`phablang`), который можно разместить в директории `/usr/bin/photon`.
- файл определения языков (`languages.def`), который должен быть установлен в той же директории, что и редактор.
- базу данных языков приложения (`myapp.ldb`).

База данных языков и файл перевода, созданные клиентом, должны размещаться:

- в одной из директорий, перечисленных в переменной окружения `ABLPATH`, если она задана.
- В той же директории, где располагается исполняемый файл, если переменная окружения `ABLPATH` не определена.

Глава 15. Контекстно-чувствительная помощь

Эта глава описывает, как обеспечить в Вашем приложении контекстно-чувствительную помощь:

- Создание текста помощи
- Ссылки на темы помощи
- Связывание помощи с виджетами
- Доступ к помощи из Вашего кода

Создание текста помощи

Чтобы создать текст помощи для чтения Photon'овским просмотрщиком помощи, Вам потребуется два типа файлов:

- сами файлы помощи
- файлы с таблицами содержания

Файлы помощи

Файлы помощи пишутся на HTML и имеют расширение .html. Просмотрщик помощи поддерживает следующие тэги (с атрибутами):

Элемент	ТЭГИ	АТРИБУТЫ
Комментарий	<!--комментарий-->	
Документ	<html>...</html>	
Заглавие	<head>...</head>	
Название	<title>...</title>	
Связь	<link>	href=usr, rel=string
Тело	<body>...</body>	
Заголовок 1	<h1>...</h1>	id=string, align={ left, center, right}
Заголовок 2	<h2>...</h2>	id=string, align={ left, center, right}
Заголовок 3	<h3>...</h3>	id=string, align={ left, center, right}
Заголовок 4	<h4>...</h4>	id=string, align={ left, center, right}
Заголовок 5	<h5>...</h5>	id=string, align={ left, center, right}
Заголовок 6	<h6>...</h6>	id=string, align={ left, center, right}
Правило	<hr>	id=string
Параграф	<p>...[</p>]	id=string
Обрыв строки	 	id=string
Образ		src=url, align={ top, middle, bottom}, alt=string, id=string (см. прим. внизу)
Анкер	<a>...	href=url, name=string, id=string
Предварительное форматирование	<pre>...</pre>	id=string
Блок ссылки	<blockquote>...</blockquote>	id=string
Адрес	<adress>...</adress>	id=string
Примечание	<note>...</note>	src=url, id=string
Список описания	<dl>...</dl>	compact, id=string
Термин	<dt>...[</dt>]	id=string
Описание	<dd>...[</dd>]	id=string
Упорядоченный список	...	id=string
Неупорядоченный список	...	id=string

Пункт списка	...	id=string
Визуальное выделение	...	id=string
Сильный	...	id=string
Код	<code>...</code>	
Образец	<samp>...</samp>	id=string
Клавиатура	<kbd>...</kbd>	id=string
Переменная	<var>...</var>	id=string
Определение	<dfn>...</dfn>	id=string
Ссылка	<cite>...</cite>	id=string
Телетайп	<tt>...</tt>	id=string
Жирный	...	id=string
Наклонный	<i>...</i>	id=string
Подчёркнутый	<u>...</u>	id=string
Таблица	<table>...</table>	border, aling={left,center,right}, id=string
Шапка таблицы	<th>...</th>	aling={left,center,right}, id=string
Данные таблицы	<td>...[</td>]	aling={left,center,right}, id=string
Строка таблицы	<tr>...[</tr>]	id=string

Просмотрщик помощи использует виджет PtWebClient и поддерживает общие форматы образов Интернета. Он также поддерживает объекты стандарта HTML 3.2/ISO для символов, а также следующее:

ОБЪЕКТ	СМЫСЛ	ОТОБРАЖАЕТСЯ КАК:
 	Фиксированный пробел	Пробел
 	Широкий пробел	Пробел
 	Узкий (нормальный) пробел	Пробел
—	Широкая черта (тире)	Штрих (-)
–	Короткая черта (дефис)	Штрих (-)
“	Левые двойные кавычки	"
”	Правые двойные кавычки	"
‘	Левые одинарные кавычки	'
’	Правые одинарные кавычки	'
™	Торговая марка	TM

Файлы таблиц содержания

Файлы таблиц содержания (Table-of-contents-TOC) определяют список продуктов, имеющих информационную помощь, и иерархию тем для каждого продукта. Эти файлы имеют расширение .toc. Все они располагаются в директории /usr/help/product. Каждый продукт имеет TOC-файл 1-го уровня и директорию с чем-либо ещё. Так, например, помощь по Photon'у включает:

```
photon.toc
photon/
```

Файл photon.toc состоит из следующей строки:

```
1 | Photon micro GUI | ./photon/bookset.html
```

где:

1 – это уровень иерархии тем;

| – разделитель областей;

Photon micro GUI – название темы.

./photon/bookset.html – унифицированный указатель информационного ресурса (URL) комплекта описания.

- ☞ Не используйте вертикальную черту (|) в названии темы, поскольку она используется в ТОС-файлах в качестве разделителя.

Директория `photon` содержит ТОС-файл и директорию для каждой книги. Например, она включает:

```
prog_guide.toc
prog_guide/
```

Файл `prog_guide.toc` похож на файл `photon.toc`:

```
2 | Programmer's Guide | ./prog_guide/about.html
```

Директория `prog_guide` содержит HTML-файлы и файл `book.toc`, который определяет названия тем в HTML-файлах:

```
3 | About This Guide | about.html#ABOUTTHISGUIDE
4 | Assumptions | #id3
4 | Chades and corrections | #ChangesAndCorrections
3 | Introduction | intro.html#id1
4 | Photon Application Builder – PhAB | #PhABApplications
6 | Get immediate results | #id3
```

.....

Часть URL, следующая за # – это анкер, определённый в HTML.

Ссылки на темы помощи

Просмотровщик помощи понимает два явных способа задания расположения HTML-ского текста помощи, который должен отображаться:

- Унифицированный указатель информационного ресурса (URL)
- Маршрут тем

Унифицированный указатель информационного ресурса (URL)

URL задаёт маршрут в файловой системе к файлу текста помощи. Он задаёт этот путь стандартным для HTML способом, за исключением того, что все файлы должны располагаться в локальной сети.

Вот пример URL:

```
/usr/help/product/photon/prog_guide/window_mgmt.html
```

- ☞ URL является чувствительным к регистру букв. Эти URL'ы ограничены рамками файлов помощи; они не могут использоваться для доступа к Всемирной Паутине.

Маршрут тем

Маршрут тем – это группа сцеплённых названий тем, определённых в дереве текущей темы. Например, вот маршрут тем, эквивалентный приведенному выше URL'у:

```
/Photon microGUI/Programmer's Guide/Window Management
```

Для просмотрщика помощи маршрут тем является нечувствительным к регистрам букв (в отличие от других просмотрщиков HTML-файлов) и маршрут может содержать групповые

символы * или ?, где "*" согласуется со строкой, и "?" согласуется с символом. Выбирается первая тема, удовлетворившая сравнению.

Дерево тем, используемое просмотрщиком помощи, должно иметь по меньшей мере три уровня иерархии: верхний уровень известен как "книжная полка", второй – как "собрание сочинений", и третий – как "книга". Книга может содержать последующие уровни – глав, разделов.

Входы в книжную полку или книгу не могут содержать каких-либо HTML-файлов, а только .toc – входы на следующий уровень; текст помощи должен находиться только в книгах.

Связывание помощи с виджетами

Вы можете отобразить информацию помощи для виджета в просмотрщике помощи либо во всплывающем "баллоне", содержащем текст помощи. Вы можете использовать даже два метода сразу в одном приложении. Например, Вы можете использовать баллон для кратких пояснений и просмотрщик помощи для более детальной помощи.

Независимо от того, какой метод Вы выбрали, Вам необходимо выполнить в каждом окне Вашего приложения следующее:

1. Установить флаг `Ph_WM_HELP` в ресурсе управления флагов (`Pt_ARG_WINDOW_MANAGER_FLAGS`).
2. Установить флаг `Ph_WM_RENDER_HELP` в ресурсе отображения флагов (`Pt_ARG_WINDOW_RENDER_FLAGS`). Это добавит иконку "?" на рамку окна. Пользователь может щёлкнуть на иконке, затем щёлкнуть на виджете, и отобразится информация с помощью.

Если для Вашего приложения не применимо по каким-либо причинам использование иконки "?", см. раздел "Помощь без иконки" ниже в этой главе.

Для более полной информации см. главу "Управление окном".

Отображение помощи в просмотрщике помощи

Чтобы использовать просмотрщик помощи для отображения информации с помощью по виджету, выполните следующее:

1. Необязательно, задайте ресурс корня помощи (`Pt_ARG_WINDOW_HELP_ROOT`) для каждого окна Вашего приложения. Это позволит Вам задать относительные маршруты тем для виджетов внутри окна.

☞ Используйте не URL, а маршруты темы.

Корневая тема должна начинаться с наклонной черты (/) и должна быть вершиной всех тем окна, обычно берётся из ТОС-файла в директории `/usr/help/product`. Например:

`/Photon micro GUI/User's Guide`

2. Для каждого виджета в окне заполните ресурс темы помощи (`Pt_ARG_HELP_TOPIC`). Если Вы задали для окна корневую тему, этот маршрут тем может быть относительно корня темы окна. Например:

`Introduction`

Когда пользователь щёлкнет на иконке "?" и выберет виджет, в просмотрщике помощи отобразится информация помощи.

☞ Если при просьбе о помощи для виджета Вы получите сообщение об ошибке, сообщающее о неверной связи ("bad link"), убедитесь в корректном маршруте тем.

Отображение помощи во всплывающем баллоне

Поместите текст, который Вы хотите отобразить в баллоне, в ресурс виджета Чтобы для отображения помогающей информации для виджета использовать баллон:

1. Поместите текст, который Вы хотите отобразить в баллоне, в ресурс виджета темы помощи (Pt_ARG_HELP_TOPIC).
2. Установите флаг Pt_INTERNAL_HELP в ресурсе расширенных флагов виджета (Pt_ARG_EFLAGS).

Когда пользователь щёлкнет на иконке "?" и затем выберет виджет, в баллоне появится помогающая информация.

Помощь без иконки "?"

Во многих приложениях иконка "?" на рамке окна не применяется. Однако Вы всё равно можете захотеть использовать для отображения помощи Photon'овский просмотрщик помощи.

Например:

- Для экранов, чувствительных к прикосновению, иконка "?" окна может оказаться слишком маленькой.
- Вы можете захотеть изменить указатель мыши на указатель помощи при нажатии клавиши.

Чтобы изменить указатель мыши на указатель помощи, перешлите менеджеру окон событие Ph_WM_HELP. Вот такой код должен находиться в ответной реакции, прикрепленный к виджету типа PtButton с надписью "Help":

```
int help_cb( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo ) {
    PhWindowEvent_t winev;

    memset( &winev, 0, sizeof(winev) );
    winev.event_f = Ph_WM_HELP;
    winev.rid = PtWidgetRid( window );
    PtForwardWindowEvent( &winev );

    return( Pt_CONTINUE );
}
```

- ☞ Для окна должен быть установлен флаг Ph_WM_HELP в ресурсе управляемых флагов (Pt_ARG_WINDOW_MANAGER_FLAGS). Вы также должны заполнить ресурс тем помощи (Pt_ARG_HELP_TOPIC) для виджетов, имеющих помощь, как обрисовано выше.

Получение доступа к помощи из Вашего программного кода

Чтобы получить доступ к помощи из кода Вашего приложения, используйте следующие функции (описанные в "Справочнике библиотечных функций Photon'a"). Они Вам не нужны, если Вы используете метод, описанный в разделе "Связывание помощи с виджетами":

PtHelpUrl()	Отображает текст помощи по URL
PtHelpUrlRoot()	Устанавливает корневой URL
PtHelpTopic()	Отображает текст помощи по маршруту тем
PtHelpTopicRoot()	Устанавливает корневую тему
PtHelpTopicTree()	Отображает текст помощи по дереву тем
PtHelpSearch()	Ищет строку
PtHelpQuit()	Закрывает просмотрщик помощи

- ☞ Функции PtHelpUrlRoot() и PtHelpTopicRoot() не сохраняют переданную строку, так что не очищайте её, пока не закончили использование корня помощи.

Глава 16. Межпроцессные связи

Приложение Photon'a не сможет всегда работать в изоляции – время от времени ему понадобится связываться с другими процессами.

В этом разделе описано:

- Коннекция (connections) [*Вводим этот термин (для отличия от термина "соединение") как относящийся уже конкретно к механизму, обеспечиваемому библиотекой Photon'a – Прим. пер.*]
- Отсылка QNX-сообщений
- Получение QNX-сообщений
- Импульсы Photon'a
- Обработка сигналов
- Другие механизмы ввода/вывода

Операционная система QNX поддерживает различные методы межпроцессных связей IPC – interprocess communication), включая:

- сообщения
- импульсы
- сигналы

Эти методы могут быть использованы в приложении Photon, если Вы будете достаточно аккуратны. Хотя лучше всего использовать коннекции Photon'a:

- Коннекторы (connectors) позволяют двум связывающимся процессам найти друг друга. Поскольку коннекторы регистрируются в Photon'e, отсутствует риск возникновения конфликта между несколькими сессиями Photon'a, запущенных на одной машине.
- Photon'овские коннекции "знают", как направить сообщение, даже если Вы имеете множество коннекций между одной и той же парой процессов. Если же Вы используете необработанные сообщения, обслуживаемые Neutrino, и процессы ввода, Вам, возможно, понадобится самим выполнять обработку.

Вот когда необработанные сообщения Neutrino [имеются в виду сообщения механизма message passing – Прим. пер.] и/или импульсы иногда могут оказаться лучшим вариантом:

- Если один из двух связывающихся процессов не является приложением Photon'a, он не может использовать библиотеку Photon'a
- Если два процесса необязательно принадлежат одной сессии Photon'a, им понадобится несколько иной способ нахождения друг друга
- Если всё, что Вам надо – это импульсы, применение коннекции является стрельбой из пушки по воробьям.

Главная петля обработки событий Photon'a, вызываемая Вашим приложением, отвечает за обработку событий Photon'a, так что вызываются самообновление виджетов и функции ответных реакций. Такая простая управляемая событиями модель программирования, используемая вместе с библиотекой виджетов Photon'a, в некотором смысле бросает вызов разработчику приложений, поскольку управляемая событиями петля исполняет безоговорочную MsgReceive() для получения событий от Photon'a. Это означает, что Вашему приложению надо быть внимательным, если оно хочет выполнить MsgReceive(), или события Photon'a могут "заблудиться" и пользовательский интерфейс может не обновиться.

Если Вам надо:

- отвечать на другие сообщения в Вашем приложении;
- исполнять ввод/вывод, используя другие механизмы (такие как, например, чтение из трубы [*Ну не пайпами же это назвать! А слово конвейер – это другое. – Прим.пер.*]);
- обрабатывать сигналы;
- отвечать на импульсы;

Вам понадобится способ подцепить Ваш код к петле обработки событий. Аналогично, Вам может понадобится суметь добавить к Вашему приложению тайм-ауты и присоединить к ним функции ответных реакций.

Коннекции

Процесс, устанавливающий коннекцию, использует объект, называемый коннектором. Коннектор является именем, которое сервер создаёт и которым владеет, и к которому клиент прикрепляет свою коннекцию. Коннектор используется только для установки коннекции.

Коннектор имеет числовой идентификатор и может также иметь имя, с ним связанное. И имя, и идентификатор являются уникальными для своей сессии Photon'a. Вот несколько примеров того, как может быть использовано имя:

- Когда сервер запускается, он создаёт коннектор с общеизвестным именем (напр., HelpViewer). Клиенты коннектируются к нему, отсылают запросы и затем отконнектируются. Если клиент при поиске имени терпит неудачу, он порождает сервер и повторяет попытку.
- Сервер создаёт безымянный коннектор и как-то отсылает идентификатор коннектора потенциальному клиенту (эту схему используют события "перетащил-и-бросил"). Этот клиент затем использует идентификатор для коннекта с сервером.
- Клиенту всегда требуется породить новый сервер для себя, даже если копия этого сервера уже запущена. Клиент создаёт "временное имя коннектора", передаёт его серверу в качестве аргумента командной строки или переменной окружения, и затем коннектится с этим сервером.

Создание коннектора

Со стороны сервера используются следующие функции:

- PtConnectorCreate()
- PtConnectorDestroy()
- PtConnectorGetId()

Со стороны клиента используются такие функции:

- PtConnectionClientDestroy()
- PtConnectionFindId()
- PtConnectionFindName()
- PtConnectionServerDestroy()
- PtConnectionTmpName()

Установка объекта коннекции

Эти функции используются для установки объекта коннекции:

- PtConnectionClientGetUserData()
- PtConnectionClientSetError()
- PtConnectionClientSetUserData()
- PtConnectionServerGetUserData()
- PtConnectionServerSetError()
- PtConnectionServerSetUserData()
- PtConnectionAddMsgHandlers()
- PtConnectionAddEventHandlers()
- PtConnectionResizeEventBuffer()

Сообщения

Эти функции используются для получения сообщений между клиентом и сервером

- PtConnectionSend(), PtConnectionSendmx()
- PtConnectionReply(), PtConnectionReplymx()

Уведомления

- PtConnectionFlush()
- PtConnectionNotify()

Локальные коннекции

Процесс может создавать коннекцию к самому себе. Поведение такой коннекции слегка отличается от поведения нормальной коннекции:

- В случае нормальной коннекции функции PtConnectionSend(), PtConnectionSendmx() и PtConnectionNotify() просто отсылают сообщение или импульс, и никогда не исполняют никакого пользовательского кода (за исключением случая, когда функция PtConnectinNotify() требует сбросить буфер, что может привести к появлению событий Photon'a, обрабатываемых нормальным образом). Но когда коннекция локальная, функции PtConnectionSend(), PtConnectionSendmx() и PtConnectionNotify() непосредственно вызывают обработчик, и как вызывающий код, так и обработчик должны предусматривать все стороны этого эффекта.
- Другим отличием является то, что обработчики вызываются из различного контекста. Обычно ответная реакция обработчика событий или обработчика сообщений вызывается из функции ввода. Поскольку функции ввода не вызываются до тех пор, пока Вы или не вернётесь в главную петлю, или не вызовете функцию PtBkgdHandlerProcess() или PtProcessEvent(), зачастую безопасным будет принимать, что обработчик не должен вызываться, пока исполняется ответная реакция. Но если коннекция локальная, обработчик вызывается непосредственно функциями PtConnectionSend(), PtConnectionSendmx() или PtConnectionNotify(), и принятые допущения должны быть соответствующим образом пересмотрены.
- Ещё одним эффектом этого является то, что если обработчик сообщений вызывает PtConnectionNotify(), клиент получает уведомление перед ответом (перед reply). Иными словами, обработчик событий клиента вызывается до того, как выполнен возврат из функций PtConnectionSend() или PtConnectionSendmx(). Если обработчик снова вызывает функцию PtConnection() или PtConnectionSendmx(), возврат вернётся с ошибкой с errno, установленным в EBUSY (так библиотечная функция защищает себя от бесконечной рекурсии). Простейшим способом обойти это является отказ от посылки уведомлений из обработчика сообщений – вместо этого уведомление можно поместить в ответе (в reply).

Асинхронные клиентские вызовы

- PtConnectionWaitForId()
- PtConnectionWaitForName()

Соглашения по именам

Вы можете определить уникальные имена для Ваших коннекторов, следуя таким соглашением по именованию:

- Имена, не содержащие слэша (/), зарезервированы за QNX Software Systems
- продукты сторонних разработчиков могут регистрировать только имена, начинающиеся с уникальной строки (напр., адрес электронной почты или имя домена), за которой следует слэш. Например: www.acme.com/dbmanager

Пример

Это приложение использует коннектор для определения того, запущен ли уже другой экземпляр приложения. Программа принимает две опции командной строки:

- e* Если другой экземпляр приложения уже выполняется, приказать ему закрыться
- f file* Если другой экземпляр приложения уже выполняется, приказать ему открыть заданный файл, в противном случае просто открыть файл.

Вот код:

```

/* Стандартные хедеры */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

/* Инструментальные хедеры */
#include <Ph.h>
#include <Pt.h>
#include <Ap.h>

/* Локальные хедеры */
#include "abimport.h"
#include "proto.h"

enum MyMsgType {
    MY_MSGTYPE_EXIT, MY_MSGTYPE_OPEN_DOC, MY_MSGTYPE_TOFRONT
};

enum MyReplyType {
    MY_REPTYPE_SUCCESS, MY_REPTYPE_BADMSG
};

struct MyMsg {
    char docname[ PATH_MAX ];
};

struct MyReply {
    enum MyReplyType status;
};

/* Обработка сообщений клиента: */
static PtConnectionMsgFunc_t msghandler;

static void const *msghandler(
    PtConnectionServer_t *connection, void *data,
    unsigned long type, void const *msgptr,
    unsigned msglen, unsigned *reply_len
    ) {
    struct MyMsg const *msg = (struct MyMsg const*) msgptr;
    struct MyReply reply;
    reply.status = MY_REPTYPE_SUCCESS;
    switch ( type ) {
        case MY_MSGTYPE_EXIT :
            PtConnectionReply( connection, sizeof(reply), &reply );
            PtExit( EXIT_SUCCESS );
            break;
        case MY_MSGTYPE_OPEN_DOC :
            reply.status = OpenNewDocument( msg->docname );
            break;
        case MY_MSGTYPE_TOFRONT : break;
        default : reply.status = MY_REPTYPE_BADMSG;
    }
    // switch(type)
    PtWindowToFront( ABW_base );
    *reply_len = sizeof(reply);
    return &reply;
}
// Функции msghandler()

/* Установка нового коннектора: */
static PtConnectorCallbackFunc_t connector_callback;

static void connector_callback(
    PtConnector_t *connector,
    PtConnectionServer_t *connection,
    void *data ) {

```

```

static const PtConnectionMsgHandler_t
handlers = { 0, msghandler };
if ( PtConnectionAddMsgHandlers( connection, &handlers, 1 ) != 0 ) {
    fputs( "Unable to set up connection handler\n", stderr );
    PtConnectionServerDestroy( connection );
} } // Функции connector_callback()

/* Строка Опций приложения */
const char ApOptions[] = AB_OPTIONS "ef:"; /* Добавление Ваших опций в "" */

/* Функция инициализации приложения */
int init( int argc, char *argv[] ) {
    struct MyMsg msg;
    int opt;
    long msgtype = MY_MSGTYPE_TOFRONT;
    const char *document = NULL;
    static const char name[] = "me@myself.com/ConnectionExample";

    while ( ( opt = getopt( argc, argv, ApOptions ) ) != -1 )
        switch ( opt ) {
            case '?' : PtExit( EXIT_FAILURE );
            case 'e' : msgtype = MY_MSGTYPE_EXIT; break;
            case 'f' : document = optarg;
        }

    if ( document )
        if ( msgtype == MY_MSGTYPE_EXIT ) {
            fputs( "Вы не можете задать одновременно опции -e и -f\n", stderr );
            PtExit( EXIT_FAILURE );
        }
        else {
            msgtype = MY_MSGTYPE_OPEN_DOC;
            strncpy( msg.docname, document, sizeof( msg.docname)-1 );
        }

    while ( PtConnectorCreate( name, connector_callback, 0 ) == NULL ) {
        /* Если это вернуло неудачу, должно быть другой экземпляр приложения уже запущен */
        PtConnectionClient_t *clnt;
        if ( ( clnt = PtConnectionFindName( name, 0, 0 ) ) != 0 ) {
            struct MyReply reply;
            int result = PtConnectionSend( clnt, msgtype, &msg, &reply, sizeof( msg ), sizeof( reply ) );
            PtConnectionClientDestroy( clnt );
            if ( result == 0 ) PtExit( reply.status );
        }
    }

    /* Поскольку PtConnectorCreate() выполнен успешно,
       выполняется только один экземпляр приложения */
    if ( msgtype == MY_MSGTYPE_EXIT ) {
        fputs( "Не могу приказать ему завершиться; он и так не выполняется\n", stderr );
        PtExit( EXIT_FAILURE );
    }
    if ( document ) OpenNewDocument( document );
    return Pt_CONTINUE;
}

```

Отсылка QNX-сообщений

Приложение Photon'a может использовать `MsgSend()`, чтобы передавать сообщения другому процессу, но другому процессу надо сразу же выполнять функцию `MsgReply()`, поскольку события Photon'a не обрабатываются, пока приложение заблокировано. ("Расторопность" не является проблемой, если Ваше приложение имеет множество потоков, работающих с событиями, и Вы вызываете функцию `PtLeave()` перед и функцию `PtEnter()` после вызова функции `MsgSend()`).

Вот в качестве примера ответная реакция, которая извлекает строку из текстового виджета, отсылает её другому процессу, и отображает ответ в том же текстовом виджете:

```

/* Ответная реакция, отсылающая сообщение другому процессу */

/* Стандартные хедеры */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

```

```

#include <string.h>
#include <sys/neutrino.h>          /* Требуется для MsgSend() */

/* Инструментальные хедеры */
#include <Ph.h>
#include <Pt.h>
#include <Ap.h>

/* Локальные хедеры */
#include "globals.h"
#include "abimport.h"
#include "proto.h"

extern int coid;

int send_msg_to_b( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo ) {
char *a_message;

/* предотвращает предупреждения (варнинги) об отсутствии ссылок */
widget = widget, apinfo = apinfo, cbinfo = cbinfo;

/* Получает строку из текстового виджета */
PtGetResource (ABW_msg_text, Pt_ARG_TEXT_STRING, 0, 0);

/* Отсылает строку другому процессу */
a_message = (char *)args[0].value;
if ( MsgSend (coid, a_message, msg_size, rcv_msg, msg_size) == -1) {
    perror ("Отсылка не удалась ");
    PtExit (-1);
}

/* Помните, что UI (пользовательский интерфейс) "висит",
пока другой процесс не ответит! */

/* Отображение ответа в том же текстовом виджете */

PtSetResource (ABW_msg_text, Pt_ARG_TEXT_STRING, rcv_msg, 0);

return( Pt_CONTINUE );
}

```

Для получения подробной информации см. "Руководство по системной архитектуре QNX 6".

Приём QNX-сообщений

Чтобы получать события от Photon, библиотечные функции виджета выполняют безоговорочный `MsgReceive()`, помещающий принятое событие в буфер событий контекста приложения. Если сообщение не является событием Photon'a, оно отвергается, пока Вы не зарегистрировали в Вашем приложении некую процедуру обработки ввода (или обработчик ввода).

☞ Вы можете создать Ваш собственный канал и вызвать функцию `MsgReceive()` в нём, но помните, что Ваше приложение и его интерфейс будут заблокированы до тех пор, пока процесс не пришлёт сообщение. Лучше использовать обработчик ввода, как это описано в настоящем разделе.

Обработчик ввода отвечает за обработку сообщений, принятых приложением от конкретного процесса. Когда Вы регистрируете обработчик библиотечными функциями виджета, Вы идентифицируете `gcvid`, с которым соединён обработчик ввода.

Вы можете определить в Вашем приложении более одного обработчика ввода для `gcvid`, но только последний зарегистрированный будет вызываться библиотекой виджета, когда от процесса будет получено сообщение.

Вы можете зарегистрировать неопределённый обработчик ввода, определив в качестве `gcvid` нулевое значение. Такой обработчик вызывается, когда приложение получает:

- любое не-Photon'овское сообщение, которое не имеет обработчика ввода, конкретно связанного с ravid клиента.
- пользовательский импульс (т.е. импульс с неотрицательным кодом).

Добавление обработчика ввода

Чтобы зарегистрировать обработчик ввода, вызовите при инициализации приложения функцию PtAppAddInput(). Её синтаксис дан ниже; более подробно информацию см. в "Справочнике библиотечных функций Photon'a".

```
PtInputId_t *PtAppAddInput(
    PtAppContext_t app_context,
    pid_t pid,
    PtInputCallbackProc_t input_func,
    void *data );
```

Аргументами функции являются:

<i>app_context</i>	Адрес контекста приложения, структуры типа PtAppContext_t, управляющей всеми данными, связанными с приложением. Обычно Вы в этом аргументе передаёте NULL, так что используется контекст по умолчанию
<i>pid</i>	Либо идентификатор процесса, с чьими сообщениями имеет дело этот обработчик, либо 0, если обработчик обрабатывает сообщения от всех процессов
<i>input_func</i>	Ваш обработчик ввода, имеющий тип PtInputCallbackProc_t. Подробности см. в "Справочнике библиотечных функций Photon'a"
<i>data</i>	Дополнительные данные, передаваемые обработчику ввода

Функция PtAppAddInput() возвращает указатель на идентификатор обработчика ввода, который понадобится Вам, если Вы захотите потом удалить этот обработчик ввода.

Прототип обработчика ввода имеет следующий вид:

```
int input_proc(void *data,
               int ravid,
               void *msg,
               size_t msglen);
```

аргументами его являются:

<i>data</i>	Указатель на какие-либо дополнительные данные, которые Вы хотите передать обработчику ввода.
<i>ravid</i>	Идентификатор отправителя – процесса, пославшего сообщение.
<i>msg</i>	Указатель на отосланное сообщение.
<i>msglen</i>	Размер буфера сообщения. Если действительное сообщение больше буфера, загрузите оставшуюся часть сообщения вызовом MsgRead().

Вы можете также объявить обработчик ввода типа PtInputCallbackProcF_t, получив дополнительную выгоду от применения прототипа, проверяемого компилятором.

☞ Если Ваш обработчик ввода изменяет изображение, он должен вызвать функцию PtFlash(), чтобы изображение наверняка обновилось.

Обработчик ввода должен возвращать одно из следующих значений:

Pt_CONTINUE Обработчик ввода не опознал сообщение. Если имеются другие обработчики ввода, прикрепленные к тому же идентификатору процесса, вызываются они. Если отсутствуют обработчики ввода, прикрепленные к этому конкретному идентификатору процесса, или если все обработчики ввода, прикрепленные к этому конкретному идентификатору процесса, вернули Pt_CONTINUE, библиотека ищет обработчики ввода, прикрепленные к нулевому идентификатору отправителя. Если все обработчики ввода вернули Pt_CONTINUE, библиотека отвечает сообщением с кодом ENOSYS.

Pt_END	Сообщение было опознано и обработано и обработчик ввода должен быть удалён из списка. Никакие другие обработчики ввода для данного сообщения не вызывались.
Pt_HALT	Сообщение было опознано и обработано, но обработчик ввода должен оставаться в списке. Никакие другие обработчики ввода для данного сообщения не вызывались.

Функции name_attach и PtAppAddInput()

Если возможно, Вам следует использовать для установки связи с другими процессами Photon'овскую коннекцию вместо функции name_attach(). Вы не можете использовать Photon'овскую коннекцию в следующих случаях:

- Клиент, подключающийся к связи, не является приложением Photon'a.
- Клиент, подключающийся к связи, принадлежит другой сессии Photon'a.

Это касается равным образом как случая, когда Вы запускаете несколько сессий Photon'a на одной машине, так и случая, когда Ваша сессия Photon'a состоит из приложений, исполняющих на нескольких машинах и два соединяющихся процесса оказываются на различных машинах.

PtAppAddInput() и name_attach() обе пытаются создать канал с установками _NTO_CHF_COID_DISCONNECT и _NTO_CHF_DISCONNECT (см. "Справочник библиотечных функций QNX 6"). Если Ваше приложение вызывает обе функции, Вам надо позволить Photon'у использовать тот же канал, что и использует функция name_attach(), вызвав прежде всего функцию PhChannelAttach() таким образом:

```
PhChannelAttach(chid, -1, NULL);
```

перед вызовами функций name_attach() или PhAppAddInput(). Если Вы хотите создать отдельный канал для Photon'a, нет разницы, создаёте ли Вы его и передаёте его функции PhChannelAttach() до или после вызова name_attach(). Но имейте в виду, что поскольку определённые механизмы библиотеки Photon'a предполагают, что канал Photon'a имеет два установленных флага DISCONNECT, они могут неправильно работать, если это не так. Одним из таких механизмов является определение нарушенной связи (см. функции PtConnectionClientSetError() и PtConnectionServerSetError()) и всё, что зависит от этого механизма.

Удаление обработчика ввода

Чтобы удалить обработчик ввода:

- Получите от него код возврата Pt_END или
- Вызовите функцию PtAppRemoveInput(), передав её в качестве аргумента идентификатор, возвращённый функцией PtAppAddInput().

Размер буфера сообщений

Как описано выше, аргументы Вашей функции ввода включают:

msg Указатель на буфер событий, использованный для получения сообщений.
msglen Размер буфера.

Этот буфер может оказаться недостаточно велик, чтобы вместить целиком всё сообщение. Одним из способов обработки этого – выделять первые несколько байт сообщения под указание типа сообщения и отсюда определять, насколько большим оно может быть. Как только Вы узнаете размер сообщения, Вы сможете:

- Прочитать сообщение целиком, вызвав функцию MsgReadv() или
- Скопировать часть, которую Вы уже получили, в новый буфер. Получить остаток сообщения, вызвав MsgReadv(). Добавить остаток сообщения к первой части.

Альтернативным способом является установка размера буфера событий таким, чтобы он мог вместить самое большое сообщение, которое получит Ваше приложение (если Вы это знаете). Это может быть выполнено функцией `PtResizeEventMsg()`. Обычно Вы должны выполнить этот вызов до того, как предполагаете получить какие-либо сообщения.

- ☞ Функция `PtResizeEventMsg()` не уменьшит буфер сообщений меньше определённого минимального размера. Это потому, что библиотека виджета хочет продолжить функционировать.

Пример – регистрация сообщений об ошибках

Следующий фрагмент кода показывает, как неспециализированный обработчик ввода может быть использован для реагирования на сообщения от других пользователей, регистрирующих ошибки. Когда одно из таких сообщений приходит, приложение отображает содержание сообщения в многострочном текстовом виджете (Этот пример предполагает, что где-то в другом месте задекларировано `log_message`).

```
int input_proc(void *client_data, int rvid, void *msg, size_t msglen) {
    struct log_message *log = (struct log_message *)msg;

    /* Обработка только регистрирующих (log) сообщений */
    if (log->type == LOG_MSG) {
        PtWidget_t *text = (PtWidget_t *)client_data;
        struct log_message header;
        int msg_offset = offsetof(struct log_message, msg);
        int log_msglen;
        int status;

        /* Смотрим: если весь наш заголовок в буфере - это оно */
        if (msglen < msg_offset) {
            /* Читаем во всём заголовке */
            if (MsgRead(rvid, &header, msg_offset, 0) == -1) {
                status = errno;
                MsgError(rvid, status);
                return Pt_HALT;          /* отпускаем*/
            }
            log = &header;
        }

        log_msglen = msg_offset+log->msg_len;

        /* Смотрим, всё ли сообщение в буфере */

        if (msglen < log_msglen) {
            struct log_message *log_msg = (struct log_message *)alloca(log_msglen);

            /* Читаем остаток сообщения из пространства стека */

            if (log_msg == NULL || MsgRead(rvid, log_msg, log_msglen, 0) == -1) {
                status = errno;
                MsgError(rvid, status);
                return Pt_HALT;        /* отпускаем */
            }
            log = log_msg;
        }

        add_msg(text, log);
        status = 0;
        MspReply(rvid, 0, 0, 0);
    }

    return Pt_HALT;
}
```

Это приложение регистрирует функцию `input_proc()` как обработчик ввода для обработки не-Photon'овских сообщений от каких-либо других процессов. Функция `input_proc()` вначале проверяет тип пришедшего сообщения. Если обработчик ввода не является ответственным за этот тип сообщения, он немедленно возвращает управление. Это важно, поскольку будут вызваны также и какие-либо другие неспециализированные обработчики ввода, которые были зарегистрированы, и только один из них будет нести ответственность по данному сообщению.

Если тип полученного сообщения – регистрирующее сообщение, функция убеждается, что Photon прочёл в свой буфер событий сообщение целиком. Это можно определить, проверив длину сообщения, представленную как `msglen` в обработчике ввода. Если часть сообщения в буфере событий отсутствует, в памяти выделяется буфер сообщения и вызывается функция `MsgRead()`, чтобы получить сообщение целиком. Затем функция `input_proc()` вызывает функцию `add_msg()`, чтобы добавить сообщение в текстовый виджет, и выдаёт ответ на сообщение.

Когда `input_proc()` завершает свою работу, она возвращает значение `Pt_HALT`. Это указывает Photon'овской библиотеке виджета не удалять обработчик ввода.

Импульсы Photon'a

[Прим. пер. Импульсы Photon'a – это отнюдь не импульсы QNX 6, а прокси QNX 4. Именно этот механизм.]

В дополнении к синхронному обмену сообщений, Photon поддерживает импульсы. Процесс, желающий уведомить другой процесс, но не желающий при этом ожидать ответа, может использовать импульсы Photon'a. Например, сервер может использовать импульс, чтобы общаться с клиентом в ситуации, когда отсылка сообщения может оставить их обоих SEND-блокированными (и поэтому попавшими в тупик). Импульс Photon'a определяется по его отрицательному идентификатору процесса, который может быть использован в качестве аргумента `pid` функции `PtAppAddInput()`. Этот идентификатор процесса является локальным для Вашего приложения. Если Вы хотите, чтобы другой процесс прислал Вам импульс, Вы должны "взвести" импульс, используя функцию `PtPulseArm()`. Это создаёт объект типа `PtPulseMsg_t`, который может быть послан другому процессу в сообщении. Другой процесс затем будет способен посылать импульсы, вызывая функцию `MsgDeliverEvent()`.

☞ В ОС QNX версии 6 тип `PtPulseMsg_t` является структурой `sigevent`. Биты в `msg.sigev_value.sival_int`, которые соответствуют `_NOTIFY_COND_MASK`, сброшены, но могут быть установлены приложением, отсылающим импульс. Более подробно см. в описании функции `ionotify()` "Справочника библиотечных функций QNX 6".
`PtPulseArm()` (описанная в "Справочнике библиотечных функций Photon'a ") просто берёт структуру `sigevent`. Функции `PtPulseArmFd()` и `PtPulseArmPid()` существуют для совместимости с более ранними версиями ОС QNX и Photon microGUI.

Давайте посмотрим код, который Вам надо написать, чтобы поддерживать импульс в:

- Приложении Photon'a, которое получает импульсы
- Приложении Photon'a, которое выпускает импульсы

Приложение Photon'a, получающее импульсы

Это адресат импульсов Photon'a, который делает большую часть подготовительной работы. Ему следует

1. Создать импульс
2. Взвести импульс
3. Отослать сообщение об импульсе процессу, который будет этот импульс испускать
4. Зарегистрировать обработчик ввода для сообщений импульса
5. Послать импульс самому себе, если это необходимо
6. Удалить импульс, когда он больше не будет нужен

В нижеприведенных разделах обсуждается каждый шаг, и далее следует пример.

☞ Перед своим завершением процесс-получатель импульсов должен дать указание процессу, посылающему импульсы, перестать это делать.

Создание импульса

Чтобы создать импульс Photon'a, вызовите функцию `PtAppCreatePulse()`:

```
pid_t PtAppCreatePulse(PtAppContext_t app, int priority);
```

аргументами которой являются:

app Адрес контекста приложения, структуры типа `PtAppContext_t`, которая управляет всеми данными, связанными с приложением. Вам следует передавать `NULL` в качестве этого аргумента, так чтобы использовался контекст по умолчанию.

priority приоритет импульса. Если он равен `-1`, используется приоритет вызывающей программы.

`PtAppCreatePulse()` возвращает идентификатор импульса, который является отрицательным числом, но никогда не равен `-1`. Это конец импульса на стороне получателя.

Взведение импульса

Взведение импульса заполняет структуру `sigevent`, которая может использоваться в большинстве вызовов QNX-функций, принимающих этот тип в качестве аргумента.

☞ Нет никакой ошибки в том, чтобы иметь больше одного процесса-источника одного и того же импульса, несмотря на то, что получатель не будет в состоянии определить, какой процесс его послал.

Чтобы взвести импульс, вызовите функцию `PtPulseArm()`. Её прототип:

```
int PtPulseArm( PtAppContext_t app, pid_t pulse, struct sigevent *msg );
```

и аргументами являются:

app Указатель на `PtAppContext_t` структуру, определяющую текущий контекст приложения (обычно `NULL`)

pulse Импульс, созданный функцией `PtAppCreatePulse()`

msg указатель на сообщение импульса, созданное функцией. Это конец импульса на стороне отправителя, и мы должны будем отослать его этому процессу, как это описано ниже.

Эта функция возвращает указатель на идентификатор сообщения импульса, который понадобится нам позже.

Пересылка сообщения импульса испускателю импульсов

Метод, который Вы используете для отсылки сообщения импульса, зависит от процесса, который будет испускать импульсы:

- Для менеджеров ресурсов:

```
ionotify(fd, _NOTIFY_ACTION_ARM, _NOTIFY_COND_INPUT, &pulsemsg);
```

- Для любого другого типа процесса:

```
/*Создание своего собственного формата сообщения: */
msg.pulsemsg=pulsemsg;
MsgSendv(channel_id, &msg, msg_parts, &rmsg, rmsg_parts);
```

Регистрация обработчика ввода

Регистрация обработчика ввода для импульса похожа на регистрацию обработчика ввода для сообщения; см. раздел "Добавление обработчика ввода" выше в этой главе. Передайте идентификатор импульса, возвращённый функцией `PtAppCreatePulse()`, как параметр `pid` в функции `ptAppAddInput()`.

Аргумент `gsvId` для обработчика ввода не будет иметь обязательно то же значение, что и идентификатор импульса: он сопоставляет идентификатор импульса в битах, определённых в `_NOTIFY_DATA_MASK` (см. описание `ionotify()` в "Справочнике библиотечных функций QNX 6"), но остальные биты берутся из полученного импульса QNX.

Посылка импульса самому себе

Если приложению требуется отослать импульс самому себе, оно может вызвать функцию `PtAppPulseTrigger()`:

```
int PtAppPulseTrigger(PtAppContext_t app, pid_t pulse);
```

Параметрами этой функции являются структура типа `PtAppContext_t`, определяющая контекст приложения (обычно `NULL`), и идентификатор импульса, возвращённый функцией `PtAppCreatePulse()`.

Удаление импульса

Когда Ваше приложение больше не нуждается в импульсе, он может быть удалён вызовом функции `PtAppDeletePulse()`:

```
int PtAppDeletePulse(PtAppContext_t app, pid_t pulse_pid);
```

параметрами которой являются структура типа `PtAppContext_t`, определяющая контекст приложения (обычно `NULL`), и идентификатор импульса, возвращённый функцией `PtAppCreatePulse()`.

Пример – очередь сообщений

Это приложение, получающее импульсы Photon'a. Оно открывает очередь сообщений (`/dev/mqueue/testqueue` по умолчанию), устанавливает импульс, и использует функцию `mq_notify()`, чтобы отдать ему импульс, когда в очереди сообщений есть нечто для прочтения:

```
/* Стандартные хедеры */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <mqueue.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/stat.h>

/* Инструментальные хедеры */
#include <Ph.h>
#include <Pt.h>
#include <Ap.h>

/* Локальные хедеры */
#include "abimport.h"
#include "proto.h"

mqd_t mqd = -1;
struct sigevent sigev;

static void readqueue( void ) {
    static unsigned counter;
    unsigned mprio;
    ssize_t msize;
    char mbuf[ 4096 ];
    while ( ( msize = mq_receive( mqd, mbuf, sizeof(mbuf), &mprio ) ) >= 0 ) {
        char hbuf[ 40 ];
        PtTextModifyText( ABW_mtext, 0, 0, -1, hbuf,
            sprintf( hbuf, "Msg #%u (prio %d):\n", ++counter, mprio ) );
        PtTextModifyText( ABW_mtext, 0, 0, -1, mbuf, msize );
    }
}
```

```

if ( errno != EAGAIN ) perror( "mq_receive" );
} // Функции readqueue()

static int input_fun( void *data, int rcvid, void *message, size_t mbsize ) {
readqueue();
if ( mq_notify( mqd, &sigev ) == -1 ) perror( "mq_notify" );
return Pt_HALT;
} // Функции input_fun()

pid_t pulse;

/* Строка опций приложения */
const char ApOptions[] = AB_OPTIONS ""; /* Добавьте Ваши опции в "" */

int init( int argc, char *argv[] ) {
if ( ( pulse = PtAppCreatePulse( NULL, -1 ) ) == 0
    || PtAppAddInput( NULL, pulse, input_fun, NULL ) == NULL ) {
    fputs( "Инициализация не удалась\n", stderr );
    exit( EXIT_FAILURE );
}
PtPulseArm( NULL, pulse, &sigev );
/* предотвращает предупреждения (варнинги) об отсутствии ссылок */
argc = argc, argv = argv;
return( Pt_CONTINUE );
} // Функции init()

int open_queue( PtWidget_t *link_instance, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo ) {
const char *name;
PtArg_t arg;

if ( mqd >= 0 ) mq_close( mqd );

PtSetArg( &arg, Pt_ARG_TEXT_STRING, &name, 0 );
PtGetResources( ABW_qname, 1, &arg );

if ( ( mqd = mq_open( name, O_RDONLY | O_CREAT | O_NONBLOCK, S_IRUSR | S_IWUSR,
                    NULL ) ) < 0 )
perror( name );
else
if ( mq_notify( mqd, &sigev ) == -1 ) {
    perror( "mq_notify" );
    mq_close( mqd );
    mqd = -1;
}
else
readqueue();

/* предотвращает предупреждения (варнинги) об отсутствии ссылок */
link_instance = link_instance, apinfo = apinfo;
cbinfo = cbinfo;

return( Pt_CONTINUE );
} // Функции open_queue()

```

Приложение Photon, отправляющее импульсы

Приложение Photon'a, собирающееся отправлять импульсы, должно:

- Иметь обработчик ввода для сообщений от того приложения, которое собирается получать эти импульсы. Этот обработчик ввода создаётся так, как описано ранее в разделе "Получение сообщений QNX" этой главы. Ему необходимо обрабатывать сообщения, содержащие сообщение импульса, и сообщения, указывающие ему прекратить выдачу импульсов. Сохраните rcvid из сообщения, которое содержит сообщение импульса – этот идентификатор получателя понадобится Вам при отправке импульса.
- Отправлять импульсы, вызывая функцию MsgDeliverEvent().

Обработка сигналов

Если Вашему приложению необходимо обрабатывать сигналы, Вам понадобится установить обработчик сигналов. Проблема заключается в том, что Вы не можете вызвать функции Photon'a из обработчика сигнала, поскольку библиотека виджетов не является сигнало-безопасной или рентабельной (повторно входимой).

Чтобы обойти эту проблему, библиотека Photon'a включает обработчик сигнала. Вы регистрируете функцию обработки сигнала, и Photon вызывает её *после* того, как

- Photon'овский обработчик сигнала вернул управление
и
- завершилась вся обработка текущего виджета.



Обработывая сигналы таким способом, Вы не получаете строгой производительности реального времени, поскольку Ваша функция обработки сигнала не вызывается немедленно, тотчас.

Добавление функции обработки сигналов

Чтобы добавить функцию обработки сигнала, используйте функцию PtAppAddSignalProc(). Обычно Вы будете вызывать её в

- инициализирующей функции Вашего приложения
или
- установочной функции для окна.

Вам понадобится подключить хедер <signal.h>. Синтаксис функции PtAppAddSignalProc() следующий:

```
int PtAppAddSignalProc(    PtAppContext_t app,
                          sigset_t const *set,
                          PtSignalProc_t func,
                          void *data);
```

где аргументы следующие:

- app* Адрес контекста приложения, структуры типа PtAppContext_t, управляющей всеми данными, связанными с этим приложением. Задайте NULL в качестве этого аргумента, так чтобы использовался контекст по умолчанию.
- set* Указатель на набор сигналов, которые должны служить причиной вызова функции обработки сигналов. Для компоновки этого набора используйте функции sigemptyset() и sigaddset(). Более подробно см. в "Справочнике библиотечных функций QNX 6".
- func* Функция обработки сигналов. См. описание PtSignalProc_t в "Справочнике библиотечных функций Photon'a".
- data* Любые данные, передаваемые функции

Функция PtAppAddSignalProc() возвращает 0 в случае успеха или -1, если случается ошибка. Ваша функция обработки сигнала имеет следующий прототип:

```
int signalProcFunctions(int signum, void *data);
```

аргументами являются:

signum Номер обрабатываемого сигнала

data Параметр data, заданный в вызове функции PtAppAddSignalProc().

Если Вы хотите, чтобы обработчик сигнала остался установленным, верните Pt_CONTINUE. Чтобы удалить его для текущего сигнала, верните Pt_END (если функция была зарегистрирована и для других сигналов, она по-прежнему будет вызываться, если те появятся).

Удаление функции обработки сигналов

Чтобы удалить функцию обработки сигнала:

- Вызовите функцию `PtAppRemoveSignal()`, чтобы удалить одну или все существующие пары (функция обработки сигналов, данные).
- Верните `Pt_END` функцией обработки сигнала. Если функция была зарегистрирована для более чем одного сигнала, она останется установленной для остальных сигналов, кроме того, который как раз обрабатывался.

Другие механизмы ввода/вывода

Если Вашему приложению необходимо выполнить ввод/вывод, такие как чтение или запись в трубопровод, Вы должны добавить обработчик файлового дескриптора (fd handler). Обработчик файлового дескриптора – это функция, вызываемая в главной петле событий, когда заданный файловый дескриптор (fd) готов к вводу или выводу:

- Чтобы добавить обработчик файлового дескриптора в Ваше приложение, вызовите функцию `PtAppAddFd()` или `PtAppAddFdPri()`.
- Подробности о прототипе обработчика файлового дескриптора см. в описании `PtFdProc_t` в "Справочнике библиотечных функций Photon'a"
- Чтобы изменить режим, являющийся сферой интересов обработчика файлового дескриптора, вызовите функцию `PtAppSetFdMode()`.
- Чтобы удалить обработчик файлового дескриптора, верните из него `Pt_END` или вызовите функцию `PtAppRemoveFd()`.

Эти функции описаны в "Справочнике библиотечных функций Photon'a".

- ☞ Если обработчик файлового дескриптора изменяет изображение, он должен вызвать функцию `PtFlush()`, чтобы гарантировать обновление изображения.

Глава 17. Параллельные операции

В этой главе обсуждаются:

- Обзор
- Потоки
- Рабочие процедуры
- Фоновое исполнение

Обзор

Когда Вы выполняете операцию, отнимающую на исполнение много времени, выполнять её как простую ответную реакцию является не лучшей идеей. Во время исполнения ответной реакции виджеты Вашего приложения не будут восстанавливать свои повреждения и совсем не будут откликаться на действия пользователя. Вам необходимо разработать стратегию обработки очень длинных операций внутри Вашего приложения, так чтобы возвращать управление из Вашей ответной реакции как можно скорее.

Возврат управления из Вашей ответной реакции позволяет виджетам продолжить визуальное самообновление. Это также даёт некую визуальную обратную связь с пользователем, когда тот попытается что-то сделать. Если Вы не хотите, чтобы пользователь мог выполнить в это время какую-либо операцию пользовательского интерфейса, Вы должны деактивировать меню и кнопки команд. Вы можете сделать это установкой флага `Pt_BLOCKED` в ресурсе `Pt_ARG_FLAGS` виджетов окна приложения.

При распараллеливании операций Вы можете рассмотреть применение одного из нескольких различных механизмов:

- Если Вы не можете разбить операцию на отдельные части, обрабатывайте событие `Photon'a` во время выполнения операции; См. раздел "Фоновое исполнение" ниже.
- Если Вы можете разбить операцию на маленькие куски программы, Вы можете захотеть иметь функцию, которая отслеживает текущее состояние и исполняет один маленький кусок операции за раз. Вы можете затем установить виджет таймера и подсоединить его к ответной реакции, которая бы вызывала функцию каждый раз, когда таймер бы срабатывал. Или же Вы можете вызывать функцию из так называемой рабочей процедуры. Эти методы особенно эффективны для многократно повторяющихся операций, когда функция может быть исполнена один раз в итерации. См. раздел "Рабочие процедуры" ниже.
- Используйте множественные кнопки. Это требует определённой специфической обработки, поскольку библиотеки `Photon'a` не являются потокобезопасными (`thread-safe`); см. раздел "Потоки" ниже.
- Породите в ответной реакции другой процесс, и пусть имеется другой процесс, возвращающий результаты первого процесса приложению путём отсылки его сообщений. В этом случае очень важно быть в состоянии отслеживать протекание операции и давать пользователю визуальную обратную связь.

Фоновое исполнение

Если очень длительная операция не может быть легко разбита на отдельные части, и Вы не хотите использовать несколько потоков, Вы должны по меньшей мере вызвать функцию

PtBkgdHandlerProcess() для обработки событий Photon'a, так чтобы графический интерфейс пользователя не выглядел бы замороженным.

Если операция ну очень длительная, Вы можете вызвать PtBkgdHandlerProcess() внутри петли. Как часто Вам надо вызывать функцию PtBkgdHandlerProcess(), зависит от того, что делает Ваше приложение. Вы должны также найти способ, позволяющий пользователю знать, в каком состоянии ход исполнения операции.

Например, если Вы читаете большую директорию, Вы можете вызывать фоновый обработчик после чтения нескольких файлов. Если Вы открываете и обрабатываете каждый файл в директории, Вы должны вызывать PtBkgdHandlerProcess() после каждого файла.

☞ Безопасным является вызов функции PtBkgdHandlerProcess() в ответных реакциях, рабочих процедурах и процедурах ввода, но отнюдь не в "Draw"-методе виджета (см. книгу "Проектирование своих виджетов" или в функции прорисовки PtRaw). Если ответная реакция вызывает функцию PtBkgdHandlerProcess(), будьте осторожны, если можете вызвать ответную реакцию одновременно несколько раз. Если Вы не хотите обрабатывать эту рекурсию, то должны блокировать виджет(ы), связанный(ые) с ответной реакцией.

События Photon'a обрабатываются следующими функциями:

- PtBkgdHandlerProcess()
- PtFileSelection()
- PtProcessEvent()
- PtSpawnWait()

Рабочие процедуры

Рабочая процедура выполняется всякий раз, когда для Вашего приложения нет сообщений, на которые надо реагировать. В каждом цикле петли обработки событий Photon'a эта процедура вызывается, если не получены никакие сообщения (лучше, чем блокироваться на ожидании последующих сообщений функцией MsgReceive()). Эта процедура будет выполняться очень часто, так что стремитесь сохранить её как можно более короткой.

☞ Если Ваше рабочая процедура изменяет изображение, вызывайте функцию PtFlush(), чтобы гарантировать обновление изображения. См. раздел "Потоки и рабочие процедуры" ниже, если Вы пишете рабочую процедуру для многопоточной программы.

Рабочие процедуры собираются в стек; когда Вы регистрируете рабочую процедуру, она помещается наверх стека. Вызывается только рабочая процедура, расположенная в вершине стека. Если вы удаляете рабочую процедуру, находящуюся в вершине стека, вызывается эта, что расположена сразу под ней.

Рабочая процедура сама по себе является функцией ответной реакции, получающей единственный void* параметр – client_data. Этот параметр client_data представляет собой данные [вернее, указатель на данные. – Прим. пер.], которые Вы присоединяете к рабочей процедуре, когда регистрируете её в библиотеке виджета. Вы должны создать структуру данных для рабочей процедуры, которая содержит вся её информацию о состоянии, и предоставить её как client_data.

Чтобы зарегистрировать, или добавить, рабочую процедуру, вызовите функцию `PtAppAddWorkProc()`:

```
PtWorkProcId_t * PtAppAddWorkProc( PtAppContext_t app_context,
                                   PtWorkProc_t work_func,
                                   void *data );
```

параметрами которой являются:

- *app_context* – адрес контекста приложения, структура типа `PtAppContext_t`, которая управляет всеми данными, присоединёнными к этому приложению. Этот параметр должен быть задан как `NULL`, так чтобы использовался контекст по умолчанию.
- *work_func* – адрес функция ответной реакции – рабочая процедура. См. описание `PtWorkProc_t` в "Справочнике библиотечных функций Photon'a".
- *client_data* – данные, передаваемые функции при её вызове.

`PtAppAddWorkProc()` возвращает указатель на структуру типа `PtWorkProcId_t`, которая идентифицирует рабочую процедуру. Чтобы удалить рабочую процедуру, когда она уже больше не нужна, вызывайте `PtAppRemoveWorkProc()`:

```
void PtAppRemoveWorkProc( PtAppContext_t app_context,
                          PtWorkProcId_t *workProc_id );
```

передавая ей тот же контекст приложения и указатель, возвращённый функцией `PtAppAddWorkProc()`.

Реальный пример использования рабочей процедуры слишком велик, чтобы размещать его здесь, поэтому вот простой итеративный пример. Рабочая процедура отсчитывает большое число, периодически обновляя надпись, чтобы отражать ход её исполнения.

```
#include <Pt.h>

typedef struct workDialog {
    PtWidget_t *widget;
    PtWidget_t *label;
    PtWidget_t *ok_button;
} WorkDialog_t;

typedef struct countdownClosure {
    WorkDialog_t *dialog;
    int value;
    int maxvalue;
    int done;
    PtWorkProcId_t *work_id;
} CountdownClosure_t;

WorkDialog_t *create_working_dialog(PtWidget_t *parent) {
    PhDim_t dim;
    PtArg_t args[3];
    int nargs;
    PtWidget_t *window, *group;
    WorkDialog_t *dialog = (WorkDialog_t *) malloc(sizeof(WorkDialog_t));

    if (dialog) {
        dialog->widget = window = PtCreateWidget(PtWindow, parent, 0, NULL);

        nargs = 0;
        PtSetArg(&args[nargs], Pt_ARG_GROUP_ORIENTATION, Pt_GROUP_VERTICAL, 0);
        nargs++;
        PtSetArg(&args[nargs], Pt_ARG_GROUP_VERT_ALIGN, Pt_GROUP_VERT_CENTER, 0);
        nargs++;
        group = PtCreateWidget(PtGroup, window, nargs, args);

        nargs = 0;
        dim.w = 200;
        dim.h = 100;
        PtSetArg(&args[nargs], Pt_ARG_DIM, &dim, 0);
        nargs++;
        PtSetArg(&args[nargs], Pt_ARG_TEXT_STRING, "Counter: ", 0);
        nargs++;
        dialog->label = PtCreateWidget(PtLabel, group, nargs, args);

        PtCreateWidget(PtSeparator, group, 0, NULL);
```

```

nargs = 0;
PtSetArg(&nargs[nargs], Pt_ARG_TEXT_STRING, "Stop", 0);          nargs++;
dialog->ok_button = PtCreateWidget(PtButton, group, 1, args);
} // if(dialog)
return dialog;
} // create_working_dialog()

int done(PtWidget_t *w, void *client, PtCallbackInfo_t *call) {
CountdownClosure_t *closure = (CountdownClosure_t *)client;

call = call;

if (!closure->done) { PtAppRemoveWorkProc(NULL, closure->work_id); }
PtDestroyWidget(closure->dialog->widget);
free(closure->dialog);
free(closure);
return (Pt_CONTINUE);
} // done()

int count_cb(void *data) {
CountdownClosure_t *closure = (CountdownClosure_t *)data;
char buf[64];
int finished = 0;

if ( closure->value++ == 0 || closure->value % 1000 == 0 ) {
    sprintf(buf, "Counter: %d", closure->value);
    PtSetResource( closure->dialog->label, Pt_ARG_TEXT_STRING, buf, 0);
}

if ( closure->value == closure->maxvalue ) {
    closure->done = finished = 1;
    PtSetResource( closure->dialog->ok_button, Pt_ARG_TEXT_STRING, "Done", 0);
}

return finished ? Pt_END : Pt_CONTINUE;
} // count_cb()

int push_button_cb(PtWidget_t *w, void *client, PtCallbackInfo_t *call) {
PtWidget_t *parent = (PtWidget_t *)client;
WorkDialog_t *dialog;

w = w; call = call;

dialog = create_working_dialog(parent);

if (dialog) {
    CountdownClosure_t *closure = (CountdownClosure_t *) malloc(sizeof(CountdownClosure_t));

    if (closure) {
        PtWorkProcId_t *id;

        closure->dialog = dialog;
        closure->value = 0;
        closure->maxvalue = 200000;
        closure->done = 0;
        closure->work_id = id =
        PtAppAddWorkProc(NULL, count_cb, closure);

        PtAddCallback(dialog->ok_button, Pt_CB_ACTIVATE, done, closure);
        PtRealizeWidget(dialog->widget);
    } // if (closure)
} // if (dialog)
return (Pt_CONTINUE);
}

int main(int argc, char *argv[]) {
PhDim_t dim;
PtArg_t args[3];
int n;
PtWidget_t *window;
PtCallback_t callbacks[] = {{push_button_cb, NULL}};
char Helvetica14b[MAX_FONT_TAG];

if (PtInit(NULL) == -1) PtExit(EXIT_FAILURE);

dim.w = 200;
dim.h = 100;
PtSetArg(&args[0], Pt_ARG_DIM, &dim, 0);
if ((window = PtCreateWidget(PtWindow, Pt_NO_PARENT, 1, args)) == NULL)
PtExit(EXIT_FAILURE);

```

```

callbacks[0].data = window;
n = 0;
PtSetArg(&args[n++], Pt_ARG_TEXT_STRING, "Обратный отсчёт...", 0);

/* Используется шрифт Helvetica 14-пунктовый жирный, если тот доступен */

if (PfGenerateFontName("Helvetica", PF_STYLE_BOLD, 14, Helvetica14b) == NULL) {
    perror("невозможно сгенерировать имя шрифта");
}
else PtSetArg(&args[n++], Pt_ARG_TEXT_FONT, Helvetica14b, 0);
PtSetArg(&args[n++], Pt_CB_ACTIVATE, callbacks, sizeof(callbacks)/sizeof(PtCallback_t));
PtCreateWidget(PtButton, window, n, args);

PtRealizeWidget(window);

PtMainLoop();
return (EXIT_SUCCESS);
}

```

Когда нажимается кнопка, прикрепленная к ней ответная реакция создаёт рабочий диалог и добавляет рабочую процедуру, передавая в качестве параметра указатель *closure* на структуру, содержащую всю информацию, необходимую для выполнения отсчёта и очистки, когда отсчёт будет выполнен.

Параметр *closure* содержит указатель на диалог, текущее значение счётчика и значение, до которого ведётся отсчёт. Когда это значение будет достигнуто, рабочая процедура изменит надпись на кнопке диалога и подсоединит ответную реакцию, которая снесёт напроочь диалог в целом, когда кнопка будет нажата. Будучи так вынужденной, рабочая процедура вернёт `Pt_END`, чтобы быть удалённой.

Функция `done()` вызывается, если пользователь останавливает рабочую процедуру или если она завершает свою работу. Эта функция удаляет диалог, связанный с рабочей процедурой, и удаляет рабочую процедуру, если та была остановлена пользователем (т.е. она не выполнялась вплоть до завершения).

Если Вы запустите этот пример на исполнение, Вы можете обнаружить ещё одну из возможностей рабочих процедур – они вытесняют одна другую. Когда Вы добавляете новую рабочую процедуру, она вытесняет все остальные. Новая рабочая процедура отработает только один раз, пока не завершится или не будет удалена. После этого возобновится исполнение рабочей процедуры, исполнявшейся до этого. Это проиллюстрировано в вышеприведенном примере – в случае, когда пользователь нажимает кнопку "Count Down..." до того, как отсчёт завершится. Создаётся новый диалог отсчёта, и этот отсчёт будет исполняться вместо первого, пока не выполнится.

Уровень модульности этой вытесняемости – на уровне вызова функции. Когда функция ответной реакции для рабочей процедуры вернёт управление, эта рабочая процедура может быть вытеснена другой рабочей процедурой.

Потоки

Приложения Photon'a являются приложениями, управляемыми событиями и базирующимися на ответных реакциях; каждый раз, когда приходит событие, вызывается соответствующая ответная реакция и обрабатывает это событие, и затем управление возвращается в петлю событий на ожидание следующего события. В связи с такой структурой большинство приложений Photon'a являются однопоточными. Библиотека Photon'a позволяет Вам использовать потоки, но путём минимизации накладных расходов для однопоточных приложений. Библиотека Photon'a является скорее "дружественной по отношению к потокам", чем полностью потоко-безопасной, как функции `printf()` и `malloc()`, являющиеся потоко-безопасными.

- ☞ Не отменяйте поток, который может исполнять функцию библиотеки Photon'a или ответную реакцию (поскольку библиотека, возможно, должна выполнить какую-нибудь очистку, когда ответная реакция вернёт управление).

Этот раздел включает:

- Запирание библиотеки Photon'a
- Несколько потоков, обрабатывающих события
- Потоки реального времени
- Не-Photon'овские и Photon'овские потоки
- Модальные операции и потоки
- Завершение многопоточной программы
- Потоки и рабочие процедуры

Запирание библиотеки Photon'a

Вы можете использовать несколько потоков, устраивая Вашу программу таким образом, чтобы только поток, вызвавший функцию `PtInit()`, вызывал функции Photon'a, но Вы можете найти подобный подход слишком ограниченным. Библиотека Photon'a является по большей части однопоточковой, но имеет механизм, позволяющий безопасно использовать множество потоков. Этим механизмом является библиотечный замок, предоставляемый функциями `PtEnter()` и `PtLeave()`. Этот замок похож на большой взаимоисключающий семафор (мутекс), защищающий библиотеку Photon'a; только один поток может владеть замком в данный момент времени, и только этому потоку позволено выполнять вызовы библиотечных функций Photon'a. Любой другой поток, желающий выполнить вызов функции Photon'a, должен прежде выполнить вызов функции `PtEnter()`, которая блокирует его до тех пор, пока замок не станет доступным. Когда потоку больше не нужен замок, он вызывает функцию `PtLeave()`, чтобы позволить использовать библиотеку Photon'a другим потокам.

Чтобы написать Ваши не-Photon'овские потоки:

- Поставьте вызовы функций `PtEnter()` и `PtLeave()` вокруг каких-либо вызовов Photon'овских функций в этих потоках.
- Сгруппируйте вместе весь Photon'овский код, который сможете, и заключите его в одну пару вход/выход, поскольку это минимизирует количество потенциально сблокированных вызовов `PtEnter()` в Вашем коде.
- Попытайтесь убрать весь не-Photon'овский код, который может забирать время для своего исполнения, из Вашей секции вход/выход – в противном случае он может безо всяких на то оснований препятствовать другим потокам в выполнении их работы.

- ☞ Не вызывайте функцию `PtLeave()`, если Ваш поток не вызывал `PtEnter()`, или Ваше приложение может неправильно себя вести либо приведёт к аварии. Помните, что если Вы находитесь в функции ответной реакции, что-то должно вызвать `PtEnter()`, чтобы позволить Вам попасть туда.

Функция `PtLeave()` не передаёт автоматически библиотечный замок другому потоку, заблокированному внутри `PtEnter()`; другой поток становится разблокированным, но затем он должен конкурировать со всеми другими потоками, как если бы он просто вызвал `PtEnter()`. Вы должны использовать функции `PtEnter()` и `PtLeave()` вместо своего собственного мутекса, потому что когда функция `PtProcessEvent()` (которую вызывает функция `PtMainLoop()`) ожидает события, она отпирает библиотеку. Как только `PtProcessEvent()` получает некое событие, которое она может обработать, она вновь запирает библиотеку. Таким образом, Ваши не-Photon'овские потоки могут свободно получить доступ к функциям Photon'a, когда у Вас нет никаких событий для обработки.

Если Вы используете свой собственный мутекс, о котором `PtProcessEvent()` не знает, он отопрётся только тогда, когда его отопрёт Ваш код. Это означает, что только то время, когда Ваши не-Photon'овские потоки могут запереть мутекс, является временем, когда Ваше приложение

обрабатывает событие, которое вызвало одну из Ваших ответных реакций. Не-Photon'овские потоки не могут записывать мутекс, когда приложение находится в состоянии ожидания.

Несколько потоков, обрабатывающих события

Если вам требуется в Вашем приложении долго исполняемая ответная реакция, Вы можете сделать так, чтобы Ваша ответная реакция вызывала функцию `PtBkgdHandlerProcess()`, как было описано выше в этой главе. Вы можете также породить новый поток, для выполнения этой работы, вместо того чтобы делать её в ответной реакции.

Другим выбором является создание более одного Photon'овского потока, обрабатывающего события Photon'a в Вашем приложении. Вот так:

- Породить один или более дополнительных потоков, которые вызывают функцию `PtEnter()` вслед за `PtMainLoop()`. Если один из Ваших Photon'овских потоков получает некое событие, которое вызывает Вашу долго выполняющуюся ответную реакцию, остальные потоки могут принимать на себя обработку событий Photon'a.
- Вызвать функцию `PtLeave()` из ответной реакции, чтобы дать другим потокам доступ к библиотеке Photon'a.
- Не забыть вызвать функцию `PtEnter()` перед выходом из ответной реакции; код, вызвавший Вашу ответную реакцию, рассчитывает на собственный замок Photon'a, когда ответная реакция вернула управление.

☞ Отпирание библиотеки позволяет другим потокам модифицировать Ваши виджеты и глобальные переменные, пока Вы не видите, так что будьте внимательны.

Если Ваша ответная реакция позволяет другим потокам обрабатывать событие, пока она выполняет свою долго выполняющуюся операцию, имеется вероятность того, что пользователь, владеющий мышью, может нажать определённую кнопку вновь, вызывая вашу ответную реакцию до того, как та завершила выполнение своего первого вызова. Вы должны обеспечить, чтобы Ваше приложение либо корректно обрабатывало эту ситуацию, либо принять меры, чтобы такого не случилось. Вот несколько способов, как это осуществить:

- Блокировать Вашу кнопку перед тем, как ответная реакция вызовет `PtLeave()`, и разблокировать её после вызова функции `PtEnter()`
или
- Использовать флаг, указывающий повторно вызванной ответной реакции, что она уже запущена
или
- Использовать счётчик, если вы хотите подсчитать, а не просто игнорировать все дополнительные нажатия кнопки
или
- Использовать Ваш собственный мутекс или другой механизм синхронизации, чтобы гарантировать, что Вы не наступаете на свои собственные пятки (но остерегайтесь возможных тупиков).

Потоки реального времени

Не создавайте вызовов Photon'овских функций в потоках, которые должны иметь детерминированное поведение, удовлетворяющее требованиям реального времени. Сложно прогнозировать, как долго будет продолжаться блокирование на `PtEnter()`; это может забрать время у потока, который владеет замком для завершения обработки текущего события или вызова `PtLeave()`, особенно если это касается отсылки сообщения другим процессам (таким как менеджер окон).

Лучше иметь "рабочий поток", удовлетворяющий требованиям к Вашим потокам реального времени, и исполнять этот поток в его собственной секции входа/выхода. Переменная состояния – и, возможно, очередь запросов – является хорошим способом отсылки этих запросов между потоками.

Если Вы используете рабочие потоки, и Вам надо использовать переменную состояния, вызовите вместо функции `pthread_cond_wait()` функцию `PtCondWait()` и отделите мутекс. Функция `PtCondWait()` использует замок библиотеки Photon'a как мутекс и исполняет безусловный вызов `PtLeave()`, когда Вы блокируетесь, и `PtEnter()` – когда разблокируетесь.

Потоки блокированы до тех пор, пока:

- Они не получили доступ к библиотеке
- Они не получили толчок от сигнала
- Другой поток не выдал сигнал или трансляцию сигналов (broadcasts) переменной состояния.
- Другой поток не вызвал `PtExit()`, `exit()` или `_exit()`.

Функция `PtCondTimedWait()` похожа на `PtCondWait()`, но время блокирования ограничено таймаутом.

Не-Photon'овские и Photon'овские потоки

Библиотека отслеживает, какие из Ваших потоков являются Photon'овскими (читающими сообщения), а какие – не-Photon'овскими (нечитающими). Таким образом, библиотека всегда знает, сколько Ваших потоков способно получать и обрабатывать события. Эта информация в настоящее время используется только функцией `PtModaBlock()` (см. раздел "Модальные операции и потоки" ниже).

По умолчанию, поток, вызвавший функцию `PtInit()`, является читателем событий, а все остальные – нет. Но если нечитающий поток вызывает функцию `PtProcessEvent()` или `PtMainLoop()`, он автоматически становится читающим события.

☞ Photon не запускает новых потоков для Вас, если Вы завершили Photon'овские потоки. Вы также можете превратить нечитающий поток в читающий и обратно, передавая флаг в функцию `PtEnter()` или `PtLeave()`:

`Pt_EVENT_PROCESS_ALLOW` Превратить вызывающий поток в читающий события
`PtEVENT_PROCESS_PREVENT` Превратить вызывающий поток в нечитающий.

Если Вам не требуется изменять состояние потока (например, для не-Photon'овского потока, который никогда не обрабатывает никаких событий), не устанавливайте ни тот, ни другой из этих битов во флагах.

Если Вы вызываете функцию `Pt_Leave()` в ответной реакции, потому что собираетесь выполнить что-то достаточное длительное по времени, передайте функции `PtLeave()` признак `Pt_EVENT_PROCESS_PREVENT`. Это укажет библиотеке, что данный поток не собирается обрабатывать событие довольно значительный промежуток времени. Убедитесь, что передали `Pt_EVENT_PROCESS_ALLOW` функции `PtEnter()`, перед тем как выполнить возврат из ответной реакции.

Модальные операции и потоки

Модальная операция – это та, где Вам надо ожидать появления какого-то конкретного события перед тем, как Вы можете начать исполнение – например, когда Вы хотите, чтобы пользователь принял решение и нажал кнопку "Да" или "Нет". Поскольку обычно до того, как появится ожидаемое, обычно придут другие события, Вам надо гарантировать, что они обработаны.

В однопоточном приложении прикрепите ответную реакцию к кнопкам "Да" и "Нет". В этой ответной реакции вызовите `PtModalUnblock()`. Когда Вы отобразите диалог, вызовите функцию `PtModalBlock()`. Эта функция запустит петлю обработки событий, похожую на `PtMainLoop()`, за исключением того, что функция `PtModalBlock()` возвращает управление, когда что-нибудь (например, ответная реакция, прикрепленная к кнопкам "Да" и "Нет"), вызовет `PtModalUnblock()`.

В многопоточном приложении функция `PtModalBlock()` может:

- делать то же, что и однопоточном приложении
или
- блокироваться на переменной состояния и позволить другим потокам Photon'a обрабатывать события.

По умолчанию функция `PtModalBlock()` использует переменную состояния, если у Вас имеются какие-либо другие Photon'овские потоки. Она удаляет поток из пула обрабатываемых события потоков, но предотвращает ситуацию, когда запущенная вторая модальная операция в потоке, который запустил петлю в `PtModalBlock()`, делает невозможным для первой `PtModalBlock()` вернуть управление до тех пор, пока вторая модальная операция не будет завершена.

В большинстве приложений этого не должно произойти; обычно Вы либо не хотите позволить другие модальные операции до тех пор, пока выполняющаяся в текущий момент не завершится, либо же Вы действительно хотите получить стековый характер действий, когда вторая модальная операция не допускает завершения первой. Например, если первая модальная операция – это файловый селектор, и вторая – это вопрос "Вы уверены, что хотите переписать этот файл?", Вы не хотите позволить пользователю закрыть файловый селектор до ответа на вопрос.

Если Вы знаете, что Ваше приложение не имеет двух несвязанных модальных операций, которые могут выполняться одновременно, но могут завершаться в любом порядке, Вы можете передать признак `Pt_EVENT_PROCESS_ALLOW` в функцию `PtModalBlock()`. Это указывает функции `PtModalBlock()` запустить петлю событий, даже если Вы имеете другие доступные Photon'овские потоки, и может уменьшить общее число Photon'овских потоков, которые нужны Вашему приложению.

Завершение многопоточной программы

Завершение многопоточного приложения может оказаться мудрёным; вызов функции `exit()` сделает так, что Ваши потоки просто исчезнут, так что Вам следует убедиться, что Вы не завершите работу, пока другой поток делает что-то такое, что нельзя прервать, например, сохраняет файл.

- ☞ Не вызывайте `pthread_exit()` в потоке, который запер библиотеки Photon'a. Если вы сделаете это, в Вашем приложении будет утечка памяти.
Помните, что все ответные реакции выполняются потоком, который запер библиотеки.

В приложении Photon'a библиотека может вызвать функцию `PtExit()`, когда закрывается последнее окно Вашего приложения. Если вы не хотите, чтобы это случилось, пока поток выполняет что-нибудь важное, сбросьте признак `Ph_WM_CLOSE` в ресурсе `Pt_ARG_WINDOW_MANAGER_FLAGS` Вашего базового окна и обработайте сообщение о закрытии самостоятельно. Вам также необходимо найти все вызовы `exit()` или `PtExit()` в Вашем программном коде и принять меры, чтобы Вы не завершили работу до тех пор, пока это не станет безопасным. Если виджет в Вашем базовом окне имеет ответную реакцию типа `Done` или `Cancel`, Вы также должны это обработать.

Библиотека Photon'a предлагает несколько механизмов, чтобы сделать обработку этого типа ситуации проще и безопаснее:

- Это простой счётчик, который вынуждает блокироваться функцию `PtExit()` до тех пор, пока он не станет равен нулю.
Функции, предоставляющие этот счётчик, `PtPreventExit()` и `PtAllowExit()`, являются не только потоко-безопасными, но также безопасными в смысле реального времени: они гарантируют выполнение ограниченного объёма машинного кода и никогда не генерируют инверсию приоритета.
Этот механизм считается относительно низкоуровневым и предназначен прежде всего для потоков, которые ничего не делают с функциями Photon'a (возможно, временно – т.е. пока находятся внутри секции `PtLeave()/PtEnter()`).

Основанием является то, что определённые вызовы функций Photon'a, которые обычно являются блокирующими, просто завершают вызывающий поток, если висит PtExit() (в противном случае функция PtExit() будет потенциально надолго блокировать). Это также случается, когда поток блокируется *перед* тем, как другой поток вызывает PtExit(); заблокированный поток завершается без возвращения из заблокированного вызова. Список вызовов Photon'овских функций, которые являются "летальными" после того, как другой поток вызвал PtExit(), включает попытки обработки событий, выполнение чего-либо модального, блокирования на переменной состояния с использованием функций PtCondWait() или PtCondTimedWait(), или вызовов PtEnter() или PtLeave().

- Иногда может оказаться трудным гарантировать, чтобы Ваш поток не вызывал ничего из этого после вызова PtPreventExit() – и если это делается и убивается без возможности вызова PtAllowExit(), Ваш процесс будет заперт и Вы должны его убить.

Чтобы избежать подобных ситуаций, имеется флаг Pt_DELAY_EXIT, который Вы можете передать функции PtEnter() или PtLeave(). Выполнение этого не только помешает функциям PtEnter() или PtLeave() завершить Ваш поток, когда другой поток вызовет PtExit(), но также вызовет неявно PtPreventExit(). Если Ваш поток по какой-либо причине помирает, библиотека знает, что для Вас надо вызвать PtAllowExit(). Флаг Pt_DELAY_EXIT делает Вашу ответную реакцию "сохранить файл" столь простой:

```
my_callback( ... ) {
    PtLeave( Pt_DELAY_EXIT );
    save_file();          /* Здесь Вы в безопасности... */
    PtEnter( 0 );        /* Но это может убить Вас - и это хорошо! */
}
```

Кроме того, Вы должны обеспечить, чтобы save_file() не пыталась выполнить какие-либо "летальные" вызовы. В частности, Вы не можете поднять всплывающий диалог с сообщением об ошибке, если что-то пошло неправильно. Если Вы хотите поднять всплывающий диалог, который потенциально займёт экран на минуты или часы, Вы должны сделать это перед вызовом PtExit(), например, использованием приёма с Pt_ARG_WINDOW_MANAGER_FLAGS, обсуждённого выше.

Чтобы завершить поток, который выполняет PtMainLoop(), без прекращения работы приложения в целом, вызывайте PtQuitMainLoop().

☞ Не вызывайте функцию PtMainLoop() в потоке, который не запер библиотеки Photon'a.

Если Вы вызываете PtQuitMainLoop() из главного потока Вашего приложения, приложение прекращает свою работу. Чтобы определить, находитесь ли Вы в главном потоке или нет:

- Вызовите функцию pthread_self() из функции инициализации Вашего приложения и сохраните идентификатор потока в глобальной переменной.
- Перед вызовом PtQuitMainLoop() вызовите pthread_self() вновь и сравните возвращённый идентификатор потока с глобальной переменной. Если идентификаторы потоков различны, вызов PtQuitMainLoop() будет прекращать выполнение потока, а не приложения.

Потоки и рабочие процедуры

Заметьте следующее относительно потоков и рабочих процедур:

- Если Вы прикрепляете рабочую процедуру и у Вас имеется более одного читающего [события – Прим. пер.] потока, имеется очень узкое окно, в котором может быть немедленно вызвана рабочая процедура, вместо того чтобы запускать её после того, как иссякнут события.
- Смешение потоков и рабочих процедур может привести к маленьким проблемам; если один из других потоков добавляет рабочую процедуру в то время, когда другой поток уже находится в ожидании события, рабочая процедура может быть не вызванной до тех пор, пока Вы не получите событие.

Глава 18. Необработанное рисование и мультипликация

В этой главе описывается:

- Виджет PtRaw
- Цвет
- Атрибуты рисования
- Дуга, эллипсы, многоугольники и прямоугольники
- Линии, пиксели и массивы пикселей
- Текст
- Побитовые образы (bitmaps)
- Образы (images)
- Мультипликация
- Режим рисования напрямую
- Внеэкранный видеопамять
- Поддержка альфа-сопряжения [*alpha blending – взвешенное наложение смешиваемых цветов. Прим. пер.*]
- поддержка хроматического ключа [*Chroma key – средство объявления некоторого цвета видеорисунка "прозрачным". Прим. пер.*]
- Операции расширенного растра
- Видеорежимы
- Градиенты
- Видеоверлей

Виджет PtRaw

Подпрограммы Pg библиотеки Photon'a являются функциями рисования самого низкого уровня. Они используются библиотекой виджета для прорисовки виджета. Вы можете использовать в приложении Photon'a функции Pg, но Вашему приложению придётся:

- обрабатывать все взаимодействия с пользователем;
- определять, когда прорисовка повреждена (например, когда она открывается, будучи ранее закрытой, или когда пользователь перемещает окно);
- восстанавливать прорисовку каждый раз, когда она повреждена.

☞ Вам следует всегда, когда это возможно, использовать виджеты, поскольку они делают всё вышеперечисленное автоматически.

Если Ваше приложение должно выполнять свою собственную прорисовку, Вам следует использовать виджет PtRaw. Он делает следующее:

- Сообщает приложению, что он получил повреждение
- Сбрасывает буфер рисования почти всегда, когда это необходимо (Вам придётся самим сбрасывать буфер, например, перед операцией блитирования, т.е. пересылки массива информации большого объёма. Блитирование выполняет смещение прямоугольной области Вашего рисунка на определённое расстояние; Вы можете захотеть, чтобы Ваш рисунок обновился перед тем, как это произойдёт).

Чтобы создать виджет PtRaw в PhAB, щёлкните на его иконке в палитре виджетов:



Разместите его там, где Вы хотите выполнять прорисовку. Вы можете предусмотреть для виджета PtRaw разнообразные функции; они вызываются в порядке, данном ниже, когда виджет реализуется, и затем вызываются по необходимости:

Pt_ARG_RAW_INIT_F	Функция инициализации, которая вызывается перед тем, как вычисляется пространство, занимаемое виджетом.
Pt_ARG_RAW_EXTENT_F	Будучи предусмотренной, вычисляет пространство виджета, когда тот перемещается или изменяется в размерах.
Pt_ARG_RAW_CALC_OPAQUE_F	Вычисляет список затенённых "черепиц" виджета.
Pt_ARG_RAW_CONNECT_F	Вызывается как последний этап реализации виджета, непосредственно перед тем, как создаются какие-либо требующиеся области.
Pt_ARG_RAW_DRAW_F	Выполняет прорисовку

Большую часть времени Вам будет нужно задавать только функцию рисования (см. ниже). Вы можете использовать редактор функций Photon'a (описанный в главе "Редактирование ресурсов и ответных реакций в PhAB'e") для редактирования этих ресурсов – но прежде Вы должны присвоить рисуемому виджету уникальное имя экземпляра. Вы можете также установить эти ресурсы из своего программного кода приложения; более подробно см. раздел "Ресурсы функций" в главе "Управление ресурсами в программном коде приложения". Информацию по ресурсам PtRaw'a см. в "Справочнике виджетов Photon'a".

Функция необработанного рисования

Когда Вы создаёте виджет PtRaw в PhAB'e и приметесь редактировать его функцию Pt_ARG_RAW_DRAW_F, Вы увидите предлагаемый по умолчанию код такого вида:

```
void my_raw_draw_fn( PtWidget_t *widget, PhTile_t *damage ) {
PtSuperClassDraw( PtBasic, widget, damage );
}
```

Вызов функции PtSuperClassDraw() (описанный в "Руководстве по созданию своих собственных виджетов") вызывает функцию рисования PtBasic'a, которая рисует границы необработанного виджета, заполняет виджет, и всё прочее, как задано его ресурсами. Необработанный виджет может делать всё это самостоятельно, но использование функции PtSuperClassDraw() снижает сложность функции необработанного рисования.

При обсуждении функции необработанного рисования следует отметить несколько вопросов:

- Вам необходимо знать холст (canvas) необработанного виджета.
- Начало рисуемых примитивов находится в левом верхнем углу холста родителя необработанного виджета, а не самого необработанного виджета. Вам необходимо выполнить преобразование координат.
- Необработанный виджет можно рисовать вне своего холста, но это нельзя назвать хорошей идеей. Вы должны установить в функции рисования отсечение.
- Функции прорисовки передаётся список повреждённых областей, что может использоваться для увеличения скорости восстановления.
- Для необработанных виджетов, содержание которых изменяется динамически, Вы можете определить модель, описывающую, что прорисовать. Всё это обсуждается ниже, и следом несколько примеров простых функций прорисовки.

☞ Не вызывайте функцию PtBkgdHandlerProcess() в функции рисования виджета PtRaw. Не изменяйте каким бы то ни было способом (создавая, разрушая, устанавливая ресурсы и прочая) какие бы то ни было другие виджеты в функции прорисовки необработанного виджета. Получение ресурсов из других виджетов является безопасным. Не вызывайте функцию прорисовки непосредственно из своей программы. Вместо этого повредите виджет, вызвав функцию PtDamageWidget(), и позвольте библиотеке вызвать функцию прорисовки.

Определение холста необработанного виджета

Вы можете определить холст необработанного виджета, вызвав функцию `PtCalcCanvas()` следующим образом:

```
PhRect_t  raw_canvas;
PtCalcCanvas(widget, &raw_canvas);
```

Вам понадобится это холст при выполнении каких-либо требующихся преобразований и обрезаний.

Преобразование координат

Начальной точкой для рисования примитивов является верхний левый угол холста родителя необработанного виджета. Вам, вероятно, покажется более удобным использование в качестве начальной точки верхнего левого угла холста самого необработанного виджета.

После того, как Вы определили холст необработанного виджета, Вы можете выполнить одно из следующего:

- Добавлять координаты верхнего левого угла холста необработанного виджета ко всем координатам, передаваемым в примитивы рисования. Например, чтобы нарисовать эллипс с центром в координатах (80, 60) относительно холста необработанного виджета, необходимо сделать так:

```
PhPoint_t  c1 = { 80, 60 };
PhPoint_t  r  = { 72, 52 };

c1.x += raw_canvas.ul.x;
c1.y += raw_canvas.ul.y;
PgSetFillColor(Pg_YELLOW);
PgDrawEllipse (&c1, &r, Pg_DRAW_FILL );
```

Этот метод предпочтительный.

- Вы можете установить преобразование, вызвав функцию `PgSetTranslation()` и передав ей координаты верхнего левого угла холста необработанного виджета:

```
PhPoint_t  c1 = { 80, 60 };
PhPoint_t  r  = { 72, 52 };

PgSetTranslation (&raw_canvas.ul, Pg_RELATIVE);

PgSetFillColor(Pg_YELLOW);
PgDrawEllipse (&c1, &r, Pg_DRAW_FILL );
```

☞ Убедитесь, что Вы восстановили старое преобразование, перед тем как выйти из функции рисования необработанного виджета. Вот один из способов, как это сделать:

```
/* Восстановление преобразования извлечением координат холста необработанного виджета */
raw_canvas.ul.x *= -1;
raw_canvas.ul.y *= -1;
PgSetTranslation (&raw_canvas.ul, Pg_RELATIVE);
```

Отсечение

Как уже обсуждалось выше, в функции рисования виджета возможно рисование за пределами пространства необработанного виджета, но так делать нехорошо:

- Это может загадить остальную часть интерфейса Вашего приложения
- Если необработанное рисование вне пространства необработанного виджета окажется повреждённым, но сам необработанный виджет – нет, функция прорисовки необработанного виджета не будет вызвана, и повреждения не будут исправлены.

Можно написать функцию прорисовки, так чтобы это отсечение не было нужным, но это может сделать Ваш программный код более запутанным. Например, если Вы пытаетесь писать текст, который выходит за пределы холста необработанного виджета, Вам может понадобиться прорисовать части букв. Вам также надо иметь в виду, что произойдёт, если пользователь изменит размер необработанного виджета. Намного легче использовать функцию PtClipAdd(), чтобы установить область обрезки по холсту необработанного виджета и позволить графическому драйверу ограничивать прорисовку:

```
PtClipArea(widget, &raw_canvas);
```

Перед тем как выйти из функции прорисовки, вызовите функцию PtClipRemove(), чтобы отключить область обрезки:

```
PtClipRemove();
```

Использование повреждённых черепиц (tiles)

Если выполнение Вашей функции прорисовки необработанного виджета занимает много времени, Вы можете не захотеть перерисовать весь холст, когда повреждена только его малая часть. Вы можете ускорить восстановление, используя аргумент *damage* функции прорисовки [Прим. пер. – Да и зрительно это будет получше...]. Аргумент *damage* является указателем на связанный список структур PhTile_t (см. "Справочник библиотечных функций Photon'a"), каждая из которых включает такие члены:

rect Структура PhRect_t, определяющая повреждённую область

next Указатель на следующую черепицу в списке.

Повреждённые области являются относительными от родителя необработанного виджета. По крайней мере один из них перекрывает необработанный виджет, но некоторые из них могут и не перекрывать.

Если в связанном списке находится более одной черепицы, первая из них покрывает всю область, покрываемую остальными. Можно или использовать первую черепицу и игнорировать остальные, или игнорировать первую и использовать остальные:

```
void rawDrawFunction (PtWidget_t *widget, PhTile_t *damage) {
if (damage->next != NULL) {

    /* Если имеется больше одной черепицы, пропустить первую. */

    damage = damage->next;
}

while (damage != NULL) {

    /* Проверка damage, чтобы посмотреть, надо ли делать какую-то прорисовку:
       damage->rect.ul.x, damage->rect.ul.y,
       damage->rect.lr.x, damage->rect.lr.y
    */

    ...

    damage = damage->next; /* Переход к следующей черепице. */
}
}
```

Следующие функции (описанные в "Справочнике библиотечных функций Photon'a") работают с черепицами:

PhAddMergeTiles()	Объединение двух списков черепиц, исключаящее наложение
PhClipTilings()	Обрезка одного списка черепиц другим
PhCoalesceTiles()	Комбинирование списка черепиц
PhCopyTiles()	Копирование списка черепиц
PhDeTranslateTiles()	Вычитание смещений по x и y из вершин списка черепиц
PhFreeTiles()	Возвращение списка черепиц во внутренний пул черепиц
PhGetTile()	Получение черепицы из внутреннего пула черепиц

PhIntersectTilings()	Определение пересечения двух списков черепиц
PhMergeTiles()	Удаление всех наложений из списка черепиц
PhRectsToTiles()	Создание списка черепиц из массива прямоугольников
PhSortTiles()	Сортировка списка черепиц
PhTilesToRects()	Создание массива прямоугольников из списка черепиц
PhTranslateTiles()	Добавление смещений по x и y к вершинам списка черепиц

Использование модели для более сложного рисования

Если содержание необработанного виджета является статическим, Вы можете вызывать примитивы рисования Pg непосредственно из функции необработанной прорисовки. Если содержание является динамическим, Вам понадобится определить структуру данных или модель, которая это описывает.

Структура модели зависит от Вашего приложения; функция необработанного рисования должна быть в состоянии отследить модель и отрисовать требуемую графику. Для сохранения указателя на модель используйте ресурс необработанного виджета Pt_ARG_USER_DATA или Pt_ARG_POINTER.

Примеры простых функций прорисовки PtRaw

Эта функция прорисовки рисует пару эллипсов, один из которых обрезан:

```
void my_raw_draw_fn( PtWidget_t *widget, PhTile_t *damage ) {
    PhRect_t      raw_canvas;
    PhPoint_t     c1 = { 80, 60 };
    PhPoint_t     c2 = { 30, 210 };
    PhPoint_t     r   = { 72, 52 };

    PtSuperClassDraw( PtBasic, widget, damage);
    PtCalcCanvas(widget, &raw_canvas);

    /* Установка области обрезки по холсту необработанного виджета. */
    PtClipAdd ( widget, &raw_canvas);

    /* Рисование эллипсов. */
    c1.x += raw_canvas.ul.x;
    c1.y += raw_canvas.ul.y;
    PgSetFillColor(Pg_YELLOW);
    PgDrawEllipse ( &c1, &r, Pg_DRAW_FILL);

    c2.x += raw_canvas.ul.x;
    c2.y += raw_canvas.ul.y;
    PgSetFillColor(Pg_RED);
    PgDrawEllipse ( &c2, &r, Pg_DRAW_FILL);

    /* Сброс области обрезки. */
    PtClipRemove ();
}
```

Эта функция такая же, но она устанавливает преобразование:

```
void my_raw_draw_fn( PtWidget_t *widget, PhTile_t *damage ) {
    PhRect_t      raw_canvas;
    PhPoint_t     c1 = { 80, 60 };
    PhPoint_t     c2 = { 30, 210 };
    PhPoint_t     r   = { 72, 52 };

    PtSuperClassDraw( PtBasic, widget, damage);
    PtCalcCanvas(widget, &raw_canvas);

    /* Установка области обрезки по холсту необработанного виджета. */
    PtClipAdd ( widget, &raw_canvas);

    /* Установка преобразования, так что операции рисования выполняются
    относительно холста необработанного виджета.
    */
    PgSetTranslation (&raw_canvas.ul, Pg_RELATIVE);
}
```

```

/* Рисование эллипсов. */
PgSetFillColor(Pg_YELLOW);
PgDrawEllipse ( &c1, &r, Pg_DRAW_FILL);

PgSetFillColor(Pg_RED);
PgDrawEllipse ( &c2, &r, Pg_DRAW_FILL);

/* Восстановление преобразования путём получения координат
холста необработанного виджета.
*/
raw_canvas.ul.x *= -1;
raw_canvas.ul.y *= -1;
PgSetTranslation (&raw_canvas.ul, Pg_RELATIVE);

/* Сброс области обрезки. */
PtClipRemove ();
}

```

Цвет

В микроGUI Photon'a цвета задаются типом `PgColor_t`. Библиотека и графические драйверы интерпретируют этот тип данных в соответствии с текущей моделью цветности (описанной в документации по `PgColor_t`).

Принимаемая по умолчанию модель цветности `Pg_CM_PRGB` использует 32-битное RGB (красный-зелёный-синий) представление:

Зарезервировано	Красный	Зелёный	Синий
0000 0000	r r r r r r r r	g g g g g g g g	b b b b b b b b

Макросы для наиболее часто используемых цветов определены в `<photon/Pg.h>`.

Несмотря на то, что `PgColor_t` использует 32 бита, под цвет используется только 24 бита. Такое представление называется true color. МикроGUI Photon'a – это оконная система в true color; она использует это 24-битное RGB-предоставление внутренне.

Большинство современных графических карт использует true color (24 бита) или high color (16 бит). Однако некоторые графические драйверы используют достоинства палитры в более старых картах, основанных на палитре. В "Справочнике библиотечных функций Photon'a" описаны следующие типы данных и функции, относящиеся к работе с цветом:

<code>PgAlphaValue()</code>	Получение альфа-компонента из значения цвета
<code>PgARGB()</code>	Преобразование значений альфа, красного, зелёного и синего в комбинированный формат цвета
<code>PgBackgroundShadings()</code>	Вычисление верхнего и нижнего оттенков цветов
<code>PgBlueValue()</code>	Получение синего компонента из значения цвета
<code>PgCMY()</code>	Преобразование голубого, пурпурного и жёлтого значений [<i>CMY – cyan/magenta/yellow – Прим. пер.</i>] в комбинированный формат цвета
<code>PgColorHSV_t</code>	Значение цвета HSV [Hue-Saturation-Value, т.е. Оттенок-Насыщенность-Значение. Прим. пер.]
<code>PgColorMatch()</code>	Запрашивание наилучшего совпадения цвета
<code>PgGetColorModel()</code>	Получение текущей модели цветности
<code>PgGetPalette()</code>	Запрашивание текущей палитры цветов
<code>PgGray()</code>	Генерирование оттенков серого
<code>PgGrayValue()</code>	Получение яркости цвета
<code>PgGreenValue()</code>	Получение зелёной компоненты из значения цвета
<code>PgHSV()</code>	Преобразование оттенка, насыщенности и значения в комбинированный формат цвета
<code>PgHSV2RGB()</code>	Преобразование цветов HSV в RGB

PgRedValue()	Получение красного компонента из значения цвета
PgRGB()	Преобразование значений красного, зелёного и синего в комбинированный формат цвета
PgRGB2HSV()	Преобразование RGB-цветов в HSV
PgSetColorModel()	Установка текущей модели цветности
PgSetPalette()	Установка палитры цветов

Атрибуты рисования

При выполнении необработанного рисования Вы можете устанавливать различные атрибуты, включая шрифты, палитры, цвета заполнения, цвета и стили линий и цвета текста. Установленные Вами атрибуты оказывают влияние на все операции необработанного рисования до тех пор, пока Вы не переустановите их. Например, цвет линии затрагивает все линии, пиксели и побитовые образы, которые вы рисуете, используя примитивы рисования.

- ☞ Вам нет необходимости устанавливать эти атрибуты, если Вы используете виджеты; атрибуты прорисовки устанавливаются на основании определений и ресурсов виджетов.

Общие атрибуты

Функциями, устанавливающими общие атрибуты рисования, являются:

PgDefaultMode()	Сброс режима прорисовки и маски плоскостей в значения, принимаемые по умолчанию
PgSetDrawMode()	Установка режима прорисовки
PgSetPlaneMask()	Защита видеопамати от модифицирования

Текстовые атрибуты

PgDefaultText()	Сброс текстового атрибута к его системному значению, принимаемому по умолчанию
PgSetFont()	Установка текстового шрифта
PgSetTextColor()	Установка цвета текста
PgSetTextDither()	Установка шаблона сглаживания текста
PgSetTextTransPat()	Установка прозрачности рисования
PgSetTextXORColor()	Установка цвета в XOR-прорисовке
PgSetUnderline()	Установка цветов для подчёркнутого текста

Атрибуты заполнения

Атрибуты заполнения оказывают влияние на всё рисование, которое Вы выполняете, вызывая функции примитивов, описанных в:

- Дуги, эллипсы, многоугольники и прямоугольники
- Текст
- Побитовые образы

Функциями, устанавливающими атрибуты заполнения, являются:

PgDefaultFill()	Сброс атрибута заполнения в его принимаемое по умолчанию значение
PgSetFillColor()	Установка точного цвета заполнения
PgSetFillDither()	Установка конкретного шаблона сглаживания и цветов
PgSetFillTransPat()	Установка прозрачности рисования
PgSetFillXORColor()	Установка цвета в XOR-прорисовке

Атрибуты черты (линии)

Атрибуты черты оказывают влияние на всё рисование, которое Вы выполняете, вызывая функции примитивов, описанных в

- Дуги, эллипсы, многоугольники и прямоугольники
- Линии, пиксели и массивы пикселей
- Текст
- Побитовые образы

Функциями, устанавливающими атрибуты черты, являются:

<code>PgDefaultStroke()</code>	Сброс атрибута черты в его системное принимаемое по умолчанию значение
<code>PgSetStrokeCap()</code>	Установка того, как выглядят концы линий
<code>PgSetStrokeColor()</code>	Установка цвета последующих контуров
<code>PgSetStrokeDither()</code>	Применение шаблона цвета к контурам
<code>PgSetStrokeTransPat()</code>	Использование шаблона маски к установке прозрачности рисования контуров
<code>PgSetStrokeXORColor()</code>	Использование XOR (исключающего ИЛИ) для цвета при рисовании контуров
<code>PgSetStrokeDash()</code>	Установка пунктирных линий
<code>PgSetStroJoin()</code>	Установка того, как линии соединяются
<code>PgSetStrokeWidth()</code>	Установка толщины линии
<code>PgSetStrokeFWidth()</code>	Установка толщины линии

Дуги, эллипсы, многоугольники и прямоугольники

Библиотеки Photon'a включают группу функций примитивов, которые Вы можете использовать для рисования кривых, включая:

- прямоугольники
- прямоугольники со скруглёнными углами
- прямоугольники с фасками, прямоугольники и стрелки
- многоугольники
- дуги, круги, хорды и сектора
- спэны (spans) – сложные кривые

☞ Не используйте эти примитивы рисования в интерфейсе, который использует виджеты; виджеты переотображают себя, когда повреждаются, так что всё, нарисованное поверх них, исчезнет. Чтобы отображать кривые, линии и прочая в каком-то интерфейсе:

- Создайте виджет `PtRaw` и вызовите примитивы в его функции прорисовки. См. раздел "Виджет `PtRaw`" выше в этой главе.
или
- Используйте соответствующий графический виджет. Более полная информация – в описании `PtGraphic` в "Справочнике виджетов Photon'a".

Пользуясь различными флагами примитива, Вы можете легко нарисовать контур (черта), нарисовать заполнение "внутри" (заполнение), или нарисовать сразу и то и другое как заполненный контур. Используются атрибуты заполнения и черты. Более подробно см. "Атрибуты рисования" выше в этой главе.

Чтобы:	Установите флаги в:
Заполнить примитив, используя текущие атрибуты заполнения	Pg_Draw_FILL
Очертить контур примитива, используя текущие атрибуты черты	Pg_DRAW_STROKE
Заполнить примитив и очертить его контур, используя текущие атрибуты заполнения и черты	Pg_DRAW_FILL_STROKE

mx-версии этих функций размещают адрес примитива в буфере рисования пространства данных Вашего приложения. Когда буфер рисования сбрасывается, примитив копируется в графический драйвер. Не-mx-версии копируют в буфер рисования сам примитив.

Прямоугольники

Вы можете рисовать прямоугольники, используя текущие атрибуты рисования, путём вызова функций PgDrawIRect() или PgDrawRect().

Функция PgDrawRect() использует структуру PhRect_t (см. "Справочник библиотечных функций Photon'a" для координат прямоугольника, тогда как PgDrawIRect() позволяет Вам задавать координаты отдельно. Используйте тот метод, который хотите.

В следующем примере рисуется прямоугольник, который заполнен, но не очерчен (т.е. он не имеет контура):

```
void DrawFillRect( void ) {
    PgSetFillColor( Pg_CYAN );
    PgDrawIRect( 8, 8, 152, 112, Pg_DRAW_FILL );
}
```

Если хотите, можете вместо этого использовать функцию PgDrawRect():

```
void DrawFillRect( void ) {
    PhRect_t rect = { {8, 8}, {152, 112} };

    PgSetFillColor( Pg_CYAN );
    PgDrawRect( &rect, Pg_DRAW_FILL );
}
```

В следующем примере рисуется оконтуренный незаполненный прямоугольник:

```
void DrawStrokeRect( void ) {
    PgSetStrokeColor( Pg_BLACK );
    PgDrawIRect( 8, 8, 152, 112, Pg_DRAW_STROKE );
}
```

А здесь рисуется оконтуренный заполненный прямоугольник:

```
void DrawFillStrokeRect( void ) {
    PgSetFillColor( Pg_CYAN );
    PgSetStrokeColor( Pg_BLACK );
    PgDrawIRect( 8, 8, 152, 112, Pg_DRAW_FILL_STROKE );
}
```



Заполненные и оконтуренные прямоугольники

Прямоугольники со скруглёнными углами

Прямоугольники со скруглёнными углами программируются почти таким же образом, что и прямоугольники – простым вызовом PgDrawRoundRect() с параметром PhPoint_t, указывающим в пикселях скругления углов прямоугольника.

В следующем примере рисуется чёрный скругленный прямоугольник с скруглениями углов в пять пикселей:

```
void DrawStrokeRoundRect( void ) {
    PhRect_t rect = { {20, 20}, {100, 100} };
    PhPoint_t radii = { 5, 5 };

    PgSetStrokeColor( Pg_BLACK );
    PgDrawRoundRect( &rect, &radii, Pg_DRAW_STROKE );
}
```

Прямоугольники с фасками, прямоугольники и стрелки

Функция PgDrawBevelBox() рисует прямоугольник с фасками, который представляет из себя особый тип прямоугольника:

- Если Вы устанавливаете Pg_DRAW_FILL или Pg_DRAW_FILL_STROKE в аргументе flags, область прямоугольника с фасками будет заполнена в соответствии с текущими атрибутами заполнения.
- Если Вы устанавливаете Pg_DRAW_STROKE или Pg_DRAW_FILL_STROKE в flags, верхний и левый края рисуются в соответствии с текущими атрибутами черты, а нижний и правый [в оригинале – left, а не right, что ИМНО опечатка – Прим. пер.] края рисуются дополнительным цветом, передаваемым в качестве одного из параметров.
- Имеется также параметр, позволяющий Вам установить "глубину" фаски.

Этот код рисует тёмно-серый прямоугольник с зелёными и красными фасками шириной в четыре пикселя:

```
void DrawBevelBox( void ) {
    PhRect_t r = { 8, 8, 152, 112 };
    PgSetFillColor( Pg_DGREY );
    PgSetStrokeColor( Pg_RED );
    PgDrawBevelBox( &r, Pg_GREEN, 4, Pg_DRAW_FILL_STROKE );
}
```



Прямоугольник с фасками

Вы можете вызвать функцию PgDrawBeveled(), чтобы нарисовать прямоугольник с фасками (как возможность – со срезанными или скруглёнными углами) или стрелку с фасками. Если Вы рисуете прямоугольник с прямыми углами, результат будет тем же, что и в случае PgDrawBevelBox(). Вот код, рисующий прямоугольники со срезанными и скруглёнными углами и набор стрелок:

```
void DrawBeveled() {

    PhRect_t clipped_rect = { {10, 10}, {150, 62} };
    PhRect_t rounded_rect = { {10, 67}, {150, 119} };
    PhPoint_t clipping = { 8, 8 };
    PhPoint_t rounding = { 12, 12 };

    PhRect_t rup = { {190, 20}, {230, 40} };
    PhRect_t rdown = { {190, 90}, {230, 110} };
    PhRect_t rleft = { {165, 45}, {185, 85} };
    PhRect_t rright = { {235, 45}, {255, 85} };

    /* Рисуются прямоугольники с фасками: один со срезанными,
       другой со скруглёнными углами. */

    PgSetFillColor( Pg_GREEN );
    PgSetStrokeColor( Pg_GREY );
    PgDrawBeveled( &clipped_rect, &clipping, Pg_BLACK, 2,
                  Pg_DRAW_FILL_STROKE | Pg_BEVEL_CLIP );
}
```

```

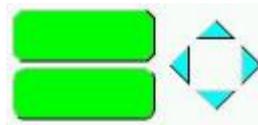
PgDrawBeveled( &rounded_rect, &rounding, Pg_BLACK, 2,
              Pg_DRAW_FILL_STROKE | Pg_BEVEL_ROUND );

/* Рисуются стрелки с фасками. */

PgSetFillColor( Pg_CYAN );
PgSetStrokeColor( Pg_GREY );

PgDrawBeveled( &rup, NULL, Pg_BLACK, 2,
              Pg_DRAW_FILL_STROKE | Pg_BEVEL_AUP );
PgDrawBeveled( &rdown, NULL, Pg_BLACK, 2,
              Pg_DRAW_FILL_STROKE | Pg_BEVEL_ADOWN );
PgDrawBeveled( &rleft, NULL, Pg_BLACK, 2,
              Pg_DRAW_FILL_STROKE | Pg_BEVEL_ALEFT );
PgDrawBeveled( &rright, NULL, Pg_BLACK, 2,
              Pg_DRAW_FILL_STROKE | Pg_BEVEL_ARIGHT );
}

```



Прямоугольники и стрелки с фасками

Если Вы хотите рисовать стрелку, встроенную в заданный прямоугольник (например, стрелку линейки протяжки), используйте функцию `PgDrawArrow()`.

Многоугольники

Вы можете создать многоугольники, задаваемого массива точек `PhPoint_t`. Если Вы используете `Pg_CLOSED` как часть аргумента `flags`, последняя точка автоматически соединяется с первой, замыкая многоугольник. Вы можете также задавать точки относительно начальной – первой точки (используя `Pg_POLY_RELATIVE`).

Следующий пример рисует синий шестиугольник с белым контуром:

```

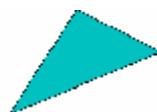
void DrawFillStrokePoly( void ) {
PhPoint_t start_point = { 0, 0 };
int num_points = 6;
PhPoint_t points[6] = {
    { 32,21 }, { 50,30 }, { 50,50 },
    { 32,59 }, { 15,50 }, { 15,30 }
};

PgSetFillColor( Pg_BLUE );
PgSetStrokeColor( Pg_WHITE );
PgDrawPolygon( points, num_points, start_point, Pg_DRAW_FILL_STROKE | Pg_CLOSED );
}

```

Перекрытие многоугольников

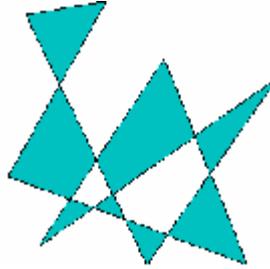
Многоугольники, которые чем-то перекрываются, заполняются с использованием так называемого правила чёт-нечет: если область пересекается чётное количество раз, она не заполнена. Чтобы понять это, давайте нарисуем горизонтальную линию, пересекающую многоугольник. Когда Вы идёте вдоль этой линии и пересекаете первую линию, Вы оказываетесь внутри многоугольника; пересекая вторую линию – выходите наружу. В качестве примера рассмотрим простой многоугольник:



Заполнение простого многоугольника

Это правило можно расширить на более сложные многоугольники:

- Когда Вы пересекаете нечётное число линий, Вы находитесь внутри многоугольника, так что область заполнена.
- Когда Вы пересекаете чётное число линий, Вы находитесь вне многоугольника, так что область не заполнена.



Заполнение перекрывающихся многоугольников

☞ Правило чёт-нечет применимо и к функции `PgDrawPoligon()` и к `PgDrawPoligonmx()`.

Дуги, круги, хорды и сектора

Функция `PgDrawArc()` может использоваться для рисования:

- дуг
- кругов
- эллипсов
- эллиптических дуг
- сегментов
- секторов

Чтобы нарисовать эллипс, можно также вызвать функцию `PgDrawEllipse()`. Начальный и конечный углы сегмента дуги задаются в бинарных градусах (*bi-grads*) – полный круг соответствует 65536 бинарным градусам (0x10000).

Чтобы нарисовать полный круг или эллипс, задайте одно и то же значение в бинарных градусах для начального и конечного углов. Например:

```
void DrawFullCurves( void ) {
  PhPoint_t circle_center = { 150, 150 }, ellipse_center = { 150, 300 };
  PhPoint_t circle_radii = { 100, 100 }, ellipse_radii = { 100, 50 };

  /* Рисование белого, незаполненного круга. */
  PgSetStrokeColor( Pg_WHITE );
  PgDrawArc( &circle_center, &circle_radii, 0, 0, Pg_DRAW_STROKE | Pg_ARC );

  /* Рисование чёрного эллипса с белым контуром. */
  PgSetFillColor( Pg_BLACK );
  PgDrawArc( &ellipse_center, &ellipse_radii, 0, 0, Pg_DRAW_FILL_STROKE | Pg_ARC );
}
```

Чтобы нарисовать сегмент (кривую, у которой крайние точки соединены прямой линией), добавьте к параметру *flags* значение `Pg_ARC_CHORD`. Например:

```
void DrawChord( void ) {
  PhPoint_t center = { 150, 150 };
  PhPoint_t radii = { 100, 50 };

  /* Рисование чёрного эллиптического сегмента с белым контуром.
  Дуга рисуется от 0 градусов до 45градусов (0x2000 биградусов). */

  PgSetStrokeColor( Pg_WHITE );
  PgSetFillColor( Pg_BLACK );
  PgDrawArc( &center, &radii, 0, 0x2000, Pg_DRAW_FILL_STROKE | Pg_ARC_CHORD );
}
```

Сходным образом, чтобы нарисовать сектор или дугу, добавьте к *flags* значения `Pg_ARC_PIE` или `Pg_ARC`. Например:

```
void DrawPieCurve( void ) {
    PhPoint_t pie_center = { 150, 150 },
              arc_center = { 150, 300 };
    PhPoint_t pie_radaii = { 100, 50 },
              arc_radaii = { 50, 100 };

    /* Рисование чёрного эллиптического сектора с белым контуром. */

    PgSetStrokeColor( Pg_WHITE );
    PgSetFillColor( Pg_BLACK );
    PgDrawArc( &pie_center, &pie_radaii, 0, 0x2000, Pg_DRAW_FILL_STROKE | Pg_ARC_PIE );

    /* Рисование чёрной дуги. */
    PgSetStrokeColor( Pg_BLACK );
    PgDrawArc( &arc_center, &arc_radaii, 0, 0x2000, Pg_DRAW_STROKE | Pg_ARC );
}
```



Заполненные и очерченные дуги

Спэны – сложные кривые

Если кривая, которую Вы хотите нарисовать, не может быть выражена как какая-либо другая кривая, поддерживаемая Photon'овским микроGUI, Вы можете нарисовать её как набор спэнов, вызвав функцию `PgDrawSpan()`. Эта функция в качестве одного из своих аргументов получает массив записей типа `PgSpan_t`.

Членами такой записи являются:

```
short x1      Начальная позиция по x1
short x2      Конечная позиция по x2
short y       Позиция по y
```

Линии, пиксели и массивы пикселей

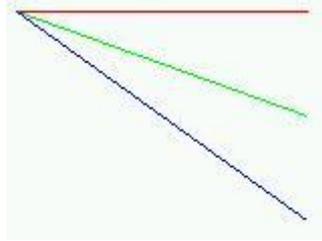
Линии и пиксели рисуются с использованием текущего состояния черты (цвет, толщина и пр.).

Примитивами рисования являются:

<code>PgDrawBezier()</code> , <code>PgDrawBeziermx()</code>	Рисование очерченных и/или заполненных кривых Безье
<code>PgDrawGrid()</code>	Рисование сетки
<code>PgDrawLine()</code> , <code>PgDrawILine()</code>	Рисование одинарной линии
<code>PgDrawPixel()</code> , <code>PgDrawIPixel()</code>	Рисование одной точки
<code>PgDrawPixelArray()</code> , <code>PgDrawPixelArraymx()</code>	Рисование множества точек
<code>PgDrawTrend()</code> , <code>PgDrawTrendmx()</code>	Рисование направленного графика (trend graph)

В следующем примере рисуются красная, зелёная и синяя линии:

```
void DrawLines( void ) {
    PgSetStrokeColor( Pg_RED );
    PgDrawILine( 8, 8, 152, 8 );
    PgSetStrokeColor( Pg_GREEN );
    PgDrawILine( 8, 8, 152, 60 );
    PgSetStrokeColor( Pg_BLUE );
    PgDrawILine( 8, 8, 152, 112 );
}
```



Линии, созданные примитивами рисования

Текст

В зависимости от Ваших потребностей можно использовать различные процедуры рисования текста:

PgDrawMultiTextArea()	Рисование в некоей области многострочного текста
PgDrawString(), PgDrawStringmx()	Рисование строки символов
PgDrawText(), PgDrawTextmx()	Рисование текста
PgDrawTextArea()	Рисование текста внутри некоей области
PgDrawTextChars()	Рисование заданного количества текстовых символов
PgExtentMultiText()	Вычисление пространства, занимаемого многострочной текстовой строкой
PgExtentText()	Вычисление пространства, занимаемого строкой текста

Текст рисуется с использованием текущих значений текстовых атрибутов; более подробно см. в разделе "Текстовые атрибуты" выше. Если Вы устанавливаете `flags` в значение `Pg_BACK_FILL`, пространство, занимаемое текстом, заполняется в соответствии с текущими атрибутами заполнения (см. "Атрибуты заполнения"). Если Вы задали подчёркивание с помощью `PgSetUnderline()`, подчёркивание рисуется под текстом и вверху фоновой заливки.

Например, чтобы напечатать чёрный текст 18-пунктовым шрифтом Helvetica:

```
void DrawSimpleText( void ) {
char *s = "Hello World!";
PhPoint_t p = { 8, 30 };
char Helvetica18[MAX_FONT_TAG];

if (PfGenerateFontName("Helvetica", 0, 18, Helvetica18) == NULL) {
    perror("Невозможно сгенерировать имя шрифта ");
}
else { PgSetFont( Helvetica18 ); }
PgSetTextColor( Pg_BLACK );
PgDrawText( s, strlen( s ), &p, 0 );
}
```

Чтобы напечатать чёрный текст на голубом фоне:

```
void DrawBackFillText( void ) {
char *s = "Hello World!";
PhPoint_t p = { 8, 30 };
char Helvetica18[MAX_FONT_TAG];

if (PfGenerateFontName("Helvetica", 0, 18, Helvetica18) == NULL) {
    perror("Невозможно сгенерировать имя шрифта");
}
else { PgSetFont( Helvetica18 ); }
PgSetTextColor( Pg_BLACK );
PgSetFillColor( Pg_CYAN );
PgDrawText( s, strlen( s ), &p, Pg_BACK_FILL );
}
```

Чтобы напечатать чёрный текст с красным подчёркиванием:

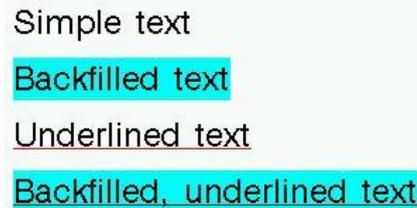
```
void DrawUnderlineText( void ) {
char *s = "Hello World!";
PhPoint_t p = { 8, 30 };
char Helvetica18[MAX_FONT_TAG];

if (PfGenerateFontName("Helvetica", 0, 18, Helvetica18) == NULL) {
    perror("Невозможно сгенерировать имя шрифта");
}
else { PgSetFont( Helvetica18 ); }
PgSetTextColor( Pg_BLACK );
PgSetUnderline( Pg_RED, Pg_TRANSPARENT, 0 );
PgDrawText( s, strlen( s ), &p, 0 );
PgSetUnderline( Pg_TRANSPARENT, Pg_TRANSPARENT, 0 );
}
```

Чтобы напечатать чёрный текст с красным подчёркиванием на голубом фоне:

```
void DrawBackFillUnderlineText( void ) {
char *s = "Hello World!";
PhPoint_t p = { 8, 30 };
char Helvetica18[MAX_FONT_TAG];

if (PfGenerateFontName("Helvetica", 0, 18, Helvetica18) == NULL) {
    perror("Невозможно сгенерировать имя шрифта");
}
else { PgSetFont( Helvetica18 ); }
PgSetTextColor( Pg_BLACK );
PgSetFillColor( Pg_CYAN );
PgSetUnderline( Pg_RED, Pg_TRANSPARENT, 0 );
PgDrawText( s, strlen( s ), &p, Pg_BACK_FILL );
PgSetUnderline( Pg_TRANSPARENT, Pg_TRANSPARENT, 0 );
}
```



Simple text
Backfilled text
Underlined text
Backfilled, underlined text

Тексты, созданные примитивами рисования

Побитовые образы (bitmaps)

Побитовые образы рисуются с использованием текущего состояния текста. Если Вы установили *flags* в `Pg_BACK_FILL`, пустые пиксели образа рисуются с использованием текущего состояния заполнения. Примитивами рисования для побитовых образов являются:

`PgDrawBitmap()`, `PgDrawBitmapx()` Рисование побитового образа
`PgDrawRepBitmap()`, `PgDrawRepBitmapx()` Рисование побитового образа несколько раз

В этом примере рисуется побитовый образ на прозрачном фоне:

```
void DrawSimpleBitmap( void ) {
PhPoint_t p = { 8, 8 };

PgSetTextColor( Pg_CELIDON );
PgDrawBitmap( TestBitmap, 0, &p, &TestBitmapSize, TestBitmapBPL, 0 );
}
```



Побитовый образ на прозрачном фоне

В этом примере рисуется побитовый образ на жёлтом фоне:

```
void DrawBackFillBitmap( void ) {
    PhPoint_t p = { 8, 8 };

    PgSetTextColor( Pg_CELIDON );
    PgSetFillColor( Pg_YELLOW );
    PgDrawBitmap( TestBitmap, Pg_BACK_FILL, &p, &TestBitmapSize, TestBitmapBPL, 0 );
}
```



Побитовый образ с затенённым фоном

Образы (images)

В этом разделе обсуждается:

- Образы на основе палитры
- Образы в непосредственных цветах
- Образы в градиентных цветах
- Создание образов
- Кеширование образов
- Прозрачность в образах
- Отображение образов
- Управление образами
- Отключение образов

Photon'овский микроGUI поддерживает следующие основные типы образов:

- В непосредственных цветах (direct color)
 - Состоит из:
 - данных образа – матрицы цветов (но необязательно типа PgColor_t). Каждый элемент в матрице является цветом пикселя.
 - Образы в непосредственных цветах имеют тип, начинающийся с Pg_IMAGE_DIRECT_
- На основе палитры
 - Состоят из:
 - палитры – массива типа PgColor_t;
 - данных образа – матрицы, элементы которой являются смещениями в палитре.
 - Образы на основе палитры имеют тип, начинающийся с Pg_IMAGE_PALETTE_
- В градиентных цветах
 - Цвета сгенерированы алгоритмически как градиент между двумя заданными цветами.

Вы можете определить любой образ через его размер пикселя [т.е. числом бит на пиксель – Прим. пер.], байтов на линию, данные образа и формат. Образ может быть сохранён в структуре типа PhImage_t (описанной в "Справочнике библиотечных функций Photon'a"). Область *type* этой структуры определяет тип образа.

Образы на основе палитры

Образы на основе палитры обеспечивают быстрый, компактный способ рисования образов. Перед тем как прорисовывать образ на основе палитры, Вы должны установить либо аппаратную, либо программную палитру, чтобы задать цвета образа. Установка аппаратной палитры изменяет физическую палитру. Весь набор цветов функции `PgSetFillColor()` выбирается из этой палитры. Другие процессы продолжают выбирать цвета из *глобальной палитры* Photon'овского микроGUI и могут выглядеть неверно. Когда Вы отключаете аппаратную палитру, остальные процессы возвращаются к нормальному отображению без перерисовки. Вы должны всегда отключать аппаратную палитру, когда Ваше окно теряет фокус.

Установка программной палитры позволяет Вам переопределить, какие цвета интерпретируются для данного рисуемого контекста *без* изменения физической палитры. Все цвета программной палитры отображаются на физическую палитру.

☞ Если Ваша физическая палитра использует больше цветов, чем поддерживает Ваша графическая карта, некоторые цвета опускаются, и образ не будет выглядеть столь красиво.

Данные образа (байты или полубайты) являются индексом в текущей палитре. Например:

```
PgColor_t      ImagePalette[256];
char          *ImageData;
PhPoint_t ImageSize;
int           ImageBPL;

void DrawYourImage( PhPoint_t pos ) {
PgSetPalette( ImagePalette, 0, 0, 256, Pg_PALSET_SOFT );
PgDrawImage( ImageData, Pg_IMAGE_PALETTE_BYTE, pos, ImageSize, ImageBPL, 0 );
}
```

Образы в непосредственных цветах

В образах в непосредственных цветах каждый пиксель может быть любого цвета. Но в сравнении с образами на основе палитры, данные образа имеют больший объём и образ, возможно, будет дольше прорисовываться. Вы можете выбирать между несколькими типами образов в непосредственных цветах, перечисленных в описании к `PhImage_t` в "Справочнике библиотечных функций Photon'a"; они отличаются размером пикселя образа и точностью цвета.

Образы в градиентных цветах

В образах в градиентных цветах цвета алгоритмически сгенерированы как градиент между двумя заданными цветами.

Создание образов

Чтобы создать структуру `PhImage_t`:

- Вызовите функцию `PhCreateImage()`
или
- Вызовите функцию `PxLoadImage()`, чтобы загрузить образ с диска
или
- Вызовите функцию `ApGetImageRes()`, чтобы загрузить образ из базы данных виджетов `PhAB'a`
или
- Получите значение ресурса `Pt_ARG_LABEL_IMAGE` виджета типа `PtLabel` или `PtButton` (поддерживаемые виджетовским `Pt_ARG_LABEL_TYPE` являются `Pt_IMAGE` или `Pt_TEXT_IMAGE`)
или
- Выделите для него место в памяти и заполните члены образа вручную.

- ☞ Лучше вызвать функцию `PhCreateImage()`, чем выделять память под структуру и заполнять её вручную. Функция `PhCreateImage()` не только предоставляет удобный способ настройки пустого образа, но также соблюдает ограничения, накладываемые графическими драйверами на выравнивание образа, и прочие вещи.

Кэширование образов

Члены `image_tag` и `palette_tag` структуры `PhImage_t` используются для кэширования образов при работе с удалёнными процессами через `phrelay` (см. "Справочник утилит QNX 6"), например, при использовании `phindows`.

Эти тэги являются контрольной суммой (CRS – т.е. полученной циклическим избыточным кодом) данных образа и палитры, и могут быть вычислены с помощью функций `PtCRC()` или `PtCRCValue()`. Если эти тэги являются ненулевыми, `phindows` и `phditto` кэшируют образы. Перед отсылкой образа `phrelay` отсылает его тэг. Если `phindows` обнаруживает тот же тэг в своём кэше, то использует образ из кэша. Эта схема уменьшает объём передаваемых данных.

- ☞ Вам нет нужды заполнять тэги, если не надо сохранять образы в кэше. Например, установите тэги в 0, если Вы отображаете мультипликацию, отображая образы, и образы никогда не повторяются.

Функции `PxLoadImage` и `ApGetImageRes()` устанавливают тэги автоматически. `PhAB` генерирует тэги для всех образов, сгенерированных через него (например, в побитовом редакторе).

Прозрачность в образах

Если Вы хотите, чтобы часть какого-то образа была прозрачной, Вы можете:

- использовать хроматический ключ
или
- создать *маску прозрачности* для образа.

Хромоключ поддерживается в большей части аппаратного обеспечения, тогда как маски прозрачности всегда обеспечиваются в программном обеспечении.

Использование хромоключа

Чтобы сделать заданный цвет прозрачным в образе, используя, если это возможно, хромоключ, вызовите функцию `PhMakeTransparent()`, передав ей образ и RGB-цвет, который Вы хотите сделать прозрачным.

Использование маски прозрачности

Маска прозрачности хранится в члене `mask_bm` структуры `PhImage_t`. Она представляет из себя побитовый образ, соответствующий данному образу; каждый бит этого побитового образа представляет пиксель рисуемого образа:

Если бит равен:	Соответствующий пиксель является:
0	Прозрачным
1	Каким-то цветом, заданным в данных образа

Член `mask_bpl` структуры `PhImage_t` задаёт число байтов на линию для маски прозрачности. Вы можете создать маску прозрачности, вызвав функцию `PhMakeTransBitmap()`.

- ☞ Если Вы используете функцию `PxLoadImage()` для загрузки прозрачного образа, установите флаг `Px_TRANSPARENT` в члене `flags` структуры `PxMethods_t`. Если Вы делаете это, функция автоматически делает образ прозрачным; Вам нет необходимости создавать маску прозрачности.

Отображение образов

Существуют различные пути отображения образов:

- Если образ хранится в структуре `PhImage_t`, вызовите функцию `PgDrawPhImage()` или `PgDrawPhImagemx()`. Эти функции автоматически обрабатывают хромоключ, альфа-операции, появление ореола на изображении, прозрачность и всё такое прочее. Чтобы прорисовывать образ периодически, вызовите функцию `PgDrawRepPhImage()` или `PgDrawRepPhImagemx()`. Чтобы прорисовать прямоугольный фрагмент образа, вызовите функцию `PgDrawPhImageRectmx()`.
- Если образ не хранится в структуре данных `PhImage_t`, вызовите функцию `PgDrawImage()` или `PgDrawImagemx()`. Чтобы прорисовать образ периодически, вызовите функцию `PgDrawRepImage()` или `PgDrawRepImagemx()`.
- Если образ не хранится в структуре `PhImage_t` и имеет маску прозрачности, вызовите функцию `PgDrawTImage()` или `PgDrawTImagemx()`.
- Установите ресурс `Pt_ARG_LABEL_IMAGE` для виджета `PtLabel` или `PtButton` (которые используют функцию `PgDrawPhImagemx()` внутренне). Ресурс `Pt_ARG_LABEL_TYPE` виджета должен быть установлен в `Pt_IMAGE` или в `Pt_TEXT_IMAGE`.

Версии `mx` этих функций размещают адрес образа в буфере рисования пространства данных Вашего приложения. Когда буфер рисования сбрасывается, весь образ копируется в графический драйвер. Не-`mx`'овские версии копируют сам образ в буфер рисования:

```
my_image-> image = PgShmenCreate (size, NULL);
```

Если Вы это делаете, данные образа не копируются в графический драйвер.

☞ Образы, созданные и возвращённые функциями `ApGetImageRes()` и `PxLoadImage()`, не размещаются в памяти совместного доступа.

Управление образами

Следующие функции позволяют Вам управлять образами:

<code>PiCropImage()</code>	Обрезает образ по заданной границе
<code>PiDuplicateImage()</code>	Дублирует образ
<code>PiFlipImage()</code>	Зеркально отображает весь образ или его часть
<code>PiGetPixel()</code>	Получает значение пикселя внутри образа
<code>PiGetPixelFromData()</code>	Получает значение из диапазона пикселей
<code>PiGetPixelRGB()</code>	Получает RGB-значение пикселя внутри образа
<code>PiSetPixel()</code>	Изменяет значение пикселя внутри образа
<code>PiSetPixelInData()</code>	Устанавливает значение пикселя в диапазоне пикселей

Освобождение образов

Структура `PhImage_t` включает член `flags`, который упрощает освобождение памяти, используемой образом. Эти флаги указывают, какие члены Вы хотите освободить:

- `Ph_RELEASE_IMAGE`
- `Ph_RELEASE_PALETTE`
- `Ph_RELEASE_TRANSPARENCY_MASK`
- `Ph_RELEASE_GHOST_BITMAP`

Вызов функции `PhReleaseImage()` с каким-то образом в качестве аргумента освобождает какие-либо ресурсы, соответствующие биту, установленному во флагах образа.

- ☞ • Функция `PhReleaseImage()` не освобождает саму структуру `PhImage_t`, а только размещённые в памяти члены её.
- Функция `PhReleaseImage()` корректно освобождает память, выделенную функцией `PgShmemCreate()`

Член `flags` для образов, созданных функциями `ApGetImageres()` и `PxLoadImage()`, не установлен. Если Вы хотите вызвать функцию `PhReleaseImage()`, чтобы освободить размещённые в памяти члены, Вы должны установить флаг самостоятельно:

```
my_image->flags = Ph_RELEASE_IMAGE |  
                 Ph_RELEASE_PALETTE |  
                 Ph_RELEASE_TRANSPARENCY_MASK |  
                 Ph_RELEASE_GHOST_BITMAP;
```

Если образ хранится в виджете, размещённые в памяти члены автоматически освобождаются, когда задаётся новый образ или виджет удаляется, обеспечивая, что соответствующие биты в члене `flags` структуры `PhImage_t` устанавливаются перед тем, как образ добавляется к виджету.

Мультипликация

В этом разделе описано, как Вы можете создавать простую мультипликацию. Здесь есть два основных шага:

- создание серии "кадров" объекта в движении
- циклический проход по кадрам

☞ Для мультипликации лучше использовать образы, а не побитовые образы, поскольку образы не являются прозрачными (при условии, что Вы не создали маску прозрачности). Это означает, что нет необходимости перерисовывать задний фон при замене одного образа на другой. В результате, когда Вы используете образы, отсутствует мерцание. Относительно других методов борьбы с мерцанием см. раздел "Мультипликация без мерцания" ниже.

Также возможно создание мультипликации использованием виджета `PtRaw` и `Photon`'овских примитивов рисования. См. раздел "Виджет `PtRaw`" выше в этой главе.

Создание серии кадров

Чтобы анимировать образ, Вам потребуется серия кадров его в движении. Например, Вы можете использовать виджет `PtLabel` (с ресурсом `Pt_ARG_LABEL_TYPE`, установленным в значение `Pt_IMAGE` или `Pt_TEXT_IMAGE`) для мультипликации. Создайте один виджет `PtLabel` там, где Вы хотите, чтобы мультипликация появилась, и создайте другой виджет `PtLabel` для каждого кадра. Вы можете разместить эти кадры в базе данных виджетов или в файле.

Использование базы данных виджетов

Как описано в разделе "Базы данных виджетов" главы "Доступ к модулям `PhAB` из программного кода", Вы можете использовать модуль картинки в качестве базы данных виджетов. Чтобы использовать его для мультипликации, выполните в `PhAB` следующее:

1. Создайте модуль картинки для его использования в качестве базы данных
2. Создайте внутреннюю связь к модулю картинки
3. Создайте кадры объекта в движении. Используйте тот же тип виджета, который Вы применяете, когда появляется анимация. Присвойте каждому кадру уникальное имя экземпляра.

В Вашей функции инициализации откройте базу данных, вызвав функцию `ApOpenDBase()`. Затем загрузите образы функций `ApGetImageRes()`. Например:

```
/* global данные */
PhImage_t *images[4];
ApDBase_t *database;
int cur_image = -1,
    num_images = 4;

int app_init( int argc, char *argv[] ) {
    int i;
    char image_name[15];

    /* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    argc = argc, argv = argv;

    database = ApOpenDBase (ABM_image_db);

    for (i = 0; i < num_images; i++) {
        sprintf (image_name, "image%d", i);
        images[i] = ApGetImageRes (database, image_name);
    }

    return (PT_CONTINUE);
}
```

☞ Функция `ApGetImageRes()` возвращает указатель в базе данных виджетов. Не закрывайте базу данных до тех пор, пока Вы используете образы из неё.

Использование файла

Вы также можете загрузить кадры из файла – из структуры `PhImage_t` – функцией `PxLoadImage()`. Эта функция поддерживает множество форматов, включая `gif`, `psx`, `jrg`, `bmp`, и `png`. Полный список см. в `/usr/photom/dll/pi_io_*`.

Циклическая прокрутка кадров

Вне зависимости от того, как Вы получили кадры, мультипликация выполняется одинаково:

1. Создайте в Вашем приложении виджет `PtTimer`. `PhAB` отображает его как чёрный прямоугольник, он не появится, когда Вы запустите своё приложение.
2. Задайте для таймера начальный (`Pt_ARG_TIMER_INITIAL`) интервал и интервалы повтора (`Pt_ARG_TIMER_REPEAT`).
3. Создайте для таймера ответную реакцию активизации (`Pt_CB_TIMER_ACTIVATE`). В ответной реакции определите следующий образ, который будет отображён, и скопируйте его в предназначенный виджет.

Например, ответная реакция таймера может быть такой:

```
/* Отображение следующего образа для нашего примера мультипликации. */

/* Стандартные хедеры */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

/* Инструментальные хедеры */
#include <Ph.h>
#include <Pt.h>
#include <Ap.h>

/* Локальные хедеры */
#include "globals.h"
#include "abimport.h"
#include "proto.h"

int display_image( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo ) {
```

```

/* предотвращает предупреждения (варнинги) об отсутствии ссылок */
widget = widget, apinfo = apinfo, cbinfo = cbinfo;

cur_image++;
if (cur_image >= num_images) cur_image=0;

PtSetResource (ABW_base_image, Pt_ARG_LABEL_IMAGE, images[cur_image], 0 );
PtFlush ();

return( Pt_CONTINUE );

}

```

ABW_base_image – это имя виджета PtLabel, в котором появляется мультипликация.

Исключение мерцания в мультипликации

Имеется два способа избежать мерцания в мультипликации:

- Создать PtOSContainer (некий контейнер внеэкрannого контекста) и сделать его родителем всех виджетов в области, где будет выполняться мультипликация
или
- Использовать функции контекстной памяти PmMem...(), чтобы собирать образ в памяти и отображать его по завершении сборки.

PtOSContainer

Когда Вы выполняете мультипликацию в виджете, родителем которого является контейнер внеэкрannого контекста, PtOSContainer формирует изображение потока прорисовки во внеэкрannой видеопамати, пользуясь преимуществами всех поддерживаемых графическим драйвером возможностей аппаратного ускорения. Графическое аппаратное обеспечение затем может битировать (т.е. копировать большой массив памяти) образ непосредственно на экран, в результате чего получаются немерцающие виджеты и/или мультипликация с отсутствием эффекта мерцания.

☞ Виджет типа PtRaw (так же, как любой другой виджет) может быть порождённым от PtOSContainer. Это означает, что Вы можете получить немерцающую мультипликацию даже тогда, когда используются примитивы рисования Photon'a.

Функции контекста памяти

Вы можете вызвать эти функции, чтобы использовать контекст памяти для уменьшения мерцания:

PmMemCreateMC()	Создать контекст памяти
PmMemFlush()	Сбросить контекст памяти в его побитовый образ
PmMemReleaseMC()	Освободить контекст памяти
PmMemSetChunkSize()	Установить величину приращения при увеличении буфера прорисовки, принадлежащего контексту памяти
PmMemSetMaxBufSize()	Установить максимальный размер буфера прорисовки, принадлежащего контексту памяти
PmMemSetType()	Установить тип контекста памяти
PmMemStart()	Сделать активным контекст памяти
PmMemStop()	Деактивировать контекст памяти

Начните с создания контекста памяти:

```

PmMemoryContext_t * PmMemCreateMC (
    PhImage_t *image,
    PhDim_t *dim,
    PhPoint_t *translation );

```

В структуре *image* должны быть заданы по меньшей мере члены *type* и *size*. Буфер данных образа является необязательным, но если Вы хотите иметь его в памяти совместного доступа, Вы должны

его обеспечить. Член *type* должен иметь значение `Pg_IMAGE_DIRECT_888` или `Pg_IMAGE_PALETTE_BYTE`.

Сразу после того, как Вы создали контекст памяти:

- Вызовите функцию `PmMemStart()`, чтобы установить контекст текущей прорисовки в контекст памяти
- Вызовите функцию `PmMemStop()`, когда закончите Вашу прорисовку, чтобы вернуться в принимаемый по умолчанию контекст прорисовки
- Вызовите функцию `PmMemFlush()`, чтобы получить результирующий образ.

Когда Вам больше не нужен контекст памяти, вызовите функцию `PmMemReleaseMC()`.

Режим рисования с прямым доступом

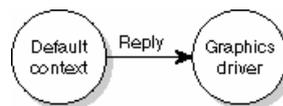
В обычном (не "напрямую") режиме, приложение отправляет запросы на прорисовку в менеджер Photon'a. Графический драйвер блокируется менеджером Photon'a.

Связь в обычном (не прямого доступа) режиме



Когда приложение входит в режим прямого доступа, оно требует, чтобы графический драйвер получал поток данных прорисовки и служебные сообщения напрямую от приложения, а не от менеджера Photon'a. Драйвер блокируется на получении данных от приложения, которое теперь ответственно за то, чтобы указывать драйверу, что делать.

Связь в режиме прямого доступа



Когда приложение входит в режим рисования с прямым доступом, приложение получает полный контроль над дисплеем, так что никакие другие приложения графическим драйвером не обслуживаются. Зона графического драйвера также больше не чувствительна к событиям рисования (так что менеджер Photon'a аннулирует все запросы других приложений к службам построения изображения этого драйвера). Другим достоинством этого режима является то, что графические услуги больше не пересылаются через пространство событий Photon'a, что улучшает производительность. Недостатком этого режима является то, что приложения, ожидающее захвата событий рисования, не могут записать визуальное представление приложения.

В целях удобства работы создан новый тип контекста `PdDirectContext_t`. Этот контекст, будучи активирован, становится контекстом по умолчанию для приложения, так что в этом режиме корректно работают вызовы всех остальных функций Photon'a `Pg*`.

☞ В этом режиме начальной точкой отсчёта для всех операций рисования является верхний левый угол экрана, поскольку запросы больше не обрезаются и не преобразуются пространством событий Photon'a. Ваше приложение, если необходимо, по-прежнему может преобразовывать и обрезать события, вызывая функции `PgSetTranslation()` и `PgSetClipping()`.

Следующие функции имеют дело с режимом рисования с прямым доступом:

PdCreateDirectContext()	Создать контекст режима рисования с прямым доступом
PdDirectStart()	Войти в режим рисования с прямым доступом
PdDirectStop()	Выйти из режима рисования с прямым доступом
PdGetDevices()	Получить идентификаторы областей (region IDs) для доступных в текущий момент устройств вывода изображения
PdReleaseDirectContext()	Выйти из режима рисования с прямым доступом и освободить контекст режима рисования с прямым доступом
PdSetTargetDevice()	Установить целевое устройство
PdWaitVSync()	Ожидать вертикальную синхронизацию

Вот несколько соображений, которые надо иметь в виду:

- Когда Вы входите или выходите из режима рисования с прямым доступом, все контексты видеопамати (за исключением дисплея) на стороне драйвера уничтожаются (драйвер генерирует событие OSINFO, так что приложение уведомляется об этом и может реинициализировать какие-либо контексты видеопамати). Это включает видеопамать, использовавшуюся структурами PdOffscreenContext_t и всё то, что использовалось видеовеерлеем программного интерфейса приложения.
- Когда Вы выходите из режима рисования с прямым доступом, на драйвер отсылается некое уведомительное сообщение, так что все другие приложения перерисовывают себя.
- Когда Вы находитесь в режиме рисования с прямым доступом, область графического драйвера больше не чувствительна к событиям рисования (так что менеджер Photon'a не накапливает огромный список событий рисования, обрабатываемых из других приложений).
- Если у Вас включено автоматическое двойное буферирование (например, devg-banshee -В...), оно отключается, пока Вы пребываете в режиме рисования с прямым доступом (чтобы позволить приложению самому управлять двойным буферированием). [Хотя здесь и множественное число: "applicationS", очевидно, что только одному приложению. Прим. пер]

Пример

Вот как получить адрес какого-либо контекста видео-памати (включая дисплей, который можно также рассматривать как видеопамать). Если Вы создали контекст прямого доступа, вызвав функцию PdCreateDirectContext(), и затем вошли в режим прямого доступа, вызвав функцию PdDirectStart(), Ваше приложение "присваивает" графический драйвер (PgFlush() проходит непосредственно на видеодрайвер, минуя сервер Photon'a).

Вам нет необходимости находится в режиме прямого доступа, чтобы получить указатель на внеэкрannую памать, но надо находиться в нём, чтобы получить указатель на основной дисплей. Вот некий псевдокод:

```

/* Создание контекста прямого доступа */
direct_context = PdCreateDirectContext();

/* Запуск режима прямого доступа */
PdDirectStart(direct_context);

/* Получение основного дисплея */
primary_display = PdCreateOffscreenContext( 0, 0, 0, Pg_OSC_MAIN_DISPLAY);

/* Получение указателя на дисплей */
vidptr = PdGetOffscreenContextPtr(primary_display);

/* Убеждаемся, что драйвер Photon'a ничего не делает
   (этого не может быть в данной точке, но мы просто убеждаемся, что
   у нас нет продвинутой движки прорисовки видеокарты)
   (??? it shouldn't be at this point but this is just to
   be sure that we haven't gotten ahead of the video card's draw engine). */

PgWaitHWIdle();

/* Делаем то, что мы делаем в памати */
Do_something(vidptr);

```

```

/* Выход из режима прямого доступа, и удаление контекста прямого доступа
   (альтернативой может быть функция PdDirectStop(),
    если мы не хотим уничтожать контекст*/
PdReleaseDirectContext(direct_mode);

```

Внеэкранный видеопамять

Эти вызовы программного интерфейса приложения позволяют Вам использовать остаточную (leftover) память видеокарты. Когда видеокарта находится в видеорежиме, это обычно остаток видеопамяти, который не используется областью дисплея. Эти области памяти могут использоваться для выполнения различных графических операций, хотя используются ещё акселератором видеокарты. В микроGUI Photon'a они в основном рассматриваются в качестве возможности, аналогичной использованию контекста памяти, но их использование будет более быстрым, поскольку для этих областей имеется аппаратное ускорение. Функции и структуры данных включают:

PdCreateOffscreenContext()	Создание внеэкранный контекста в видеопамяти
PdDupOffscreenContext()	Дублирование внеэкранный контекста
PdGetOffscreenContextPtr()	Создание ссылки на объект совместно используемой памяти внеэкранный контекста
PdOffscreenContext_t	Структура данных, описывающая внеэкранный контекст
PdSetOffscreenTranslation()	Установка преобразования для внеэкранный контекста
PdSetTargetDevice()	Установка целевого устройства
PgContextBlit()	Копирование данных из прямоугольника в одном контексте в другой контекст
PgContextBlitArea()	Копирование данных из области в одном контексте в другой контекст
PgSwapDisplay()	Указать ЭЛТ видеодисплея на заданный контекст
PgWaitHWIdle()	Ожидать, пока видеодрайвер не окажется в простое
PhDCRelease()	Освобождение контекста прорисовки

Вот некий простой код, который создаёт и формирует изображение во внеэкранный буфере, а затем копирует данные на экран. Он создаёт два внеэкранный контекста объёмом 100x100, выполняет какую-то прорисовку, блитинг, и затем блитирует результат в текущую область (т.е. в область, принадлежащую PtWindow). В этом примере предполагается, что Вы уже создали окно и вызвали функцию PgSetRegion(), чтобы указать, что область окна вызывает появление событий прорисовки:

```

#include <Pt.h>

PtInit( NULL );

/* ... */

PhDrawContext_t *olddc;
PdOffscreenContext_t *context1, *context2;
PhArea_t rsrc,rdst;

/* Создаём область 100x100 в собственном формате экрана */
context1=PdCreateOffscreenContext(0,100,100,0);
if (context1 == NULL)
{
    /* Ошибка */
}

/* Делаем её текущим контекстом и рисуем в ней пурпурный прямоугольник */
olddc=PhDCSetCurrent(context1);
PgSetFillColor(Pg_PURPLE);
PgDrawIRect(0,0,99,99,Pg_DRAW_FILL);
PgFlush(); /* Рисование прямоугольника */

PhDCSetCurrent(olddc);

```

```

/* Копируем контекст 1 и рисуем белый прямоугольник в середине пурпурного */
context2=PdDupOffscreenContext(context1,0);
if (context2==NULL)
{
    /* Ошибка */
}

olddc=PhDCSetCurrent(context2);
PgSetFillColor(Pg_WHITE);
PgDrawIRect(9,9,89,89,Pg_DRAW_FILL);
PgFlush();
PhDCSetCurrent(olddc);

/* Копируем в экранную область пурпурный прямоугольник, изображение которого
сформировано вне экрана в контексте 1 */
rsrc.pos.x = rdst.pos.x = rsrc.pos.y = rdst.pos.y = 0;
rsrc.size.w = rdst.size.w = rsrc.size.h = rdst.size.h = 100;

PgContextBlitArea(context1, &rsrc, NULL, &rdst);

/* Копируем пурпурный и белый прямоугольники из контекста 2
рядом с пурпурным на дисплей */
rdst.pos.x = 100;

PgContextBlitArea(context2, &rsrc, NULL, &rdst);
PgFlush();

/* Очищаем внеэкранный контекст */
PhDCRelease(context1);
PhDCRelease(context2);

context1=NULL;
context2=NULL;

```

Внеэкранный контекст может быть отвергнут графическим драйвером по целому ряду причин. Когда такое случается, графический драйвер отправляет менеджеру Photon'a событие `Ph_EV_INFO` с подтипом `Ph_OFFSCREEN_INVALID`. Данные этого события представляют из себя одно длинное целое, описывающее, почему внеэкранный контекст был признан недействительным. Возможными причинами являются следующие:

<code>Pg_VIDEO_MODE_SWITCHED</code>	Графический драйвер сменил видеорежимы
<code>Pg_ENTERED_DIRECT</code>	Приложение вошло в режим прямого доступа
<code>Pg_EXITED_DIRECT</code>	Приложение вышло из режима прямого доступа
<code>Pg_DRIVER_STARTED</code>	Видеодрайвер как раз начал выполнение

Приложение, которое будет использовать внеэкранные контексты, должно быть чувствительным к этим событиям и соответственно реинициализировать свои внеэкранные контексты.

Внеэкранные замки

В основном Вы будете использовать внеэкранные замки с указателями, которые Вы получили от функций `PdGetOffscreenContextPtr()`. Замки обеспечивают, что

- Команды потока рисования не рисуют, пока внеэкранный контекст заперт
- Память действенна, пока приложение её использует.

☞ Ваше приложение должно запирает внеэкранный контекст на как можно меньший промежуток времени. Если графическому драйверу требуется выполнить что-то с внеэкранный памятью, он пытается захватить замок себе, потенциально блокируя `io-graphics` на длительное время (результатом чего будет то, что экран может не обновляться, и пользователь решит, что компьютер завис).

Замки реализованы как семафоры в памяти совместного доступа между `io-graphics` и приложением.

Основными шагами использования внеэкранных замков являются:

1. Создание замка для внеэкранного контекста вызовом функции `PdCreateOffscreenLock()`. Вы можете принять меры, чтобы, если сделан запрос на удаление внеэкранного контекста, когда он заблокирован, приложению был отослан сигнал.
2. Запирание внеэкранного контекста, когда это требуется, путём вызова функции `PdLockOffscreen()`. Вы можете при желании задать таймаут блокирования.
3. Отпирание внеэкранного контекста путём вызова функции `PdUnlockOffscreen()`.
4. Когда Вам больше не нужен замок внеэкранного контекста, удаление замка вызовом функции `PdDestroyOffscreenLock()`. Когда Вы отлаживаетесь, Вы можете вызвать функцию `PdIsOffscreenLocked()`, чтобы определить, заперт ли или не заперт внеэкранный контекст в настоящий момент.

☞ Если Вы заперли контекст, вызовите функцию `PdLockOffscreen()`, чтобы отпереть его, перед тем, как удалить замок или освободить внеэкранный контекст.

Поддержка альфа-сопряжения

Альфа-сопряжение – это технология прорисовки прозрачности при рисовании какого-то объекта. В этой технологии комбинируется цвет рисуемого объекта (источник) и цвет чего-то того, на чём сверху рисуется объект (получатель). Чем больше объём цвета источника, тем более непрозрачным выглядит объект.

Математически фактор смешения представляет из себя вещественное число в диапазоне от 0 до 1 включительно. В Photon микроGUI этот фактор хранится в 8 битах, т.е. масштабирован в диапазоне от 0 до 255 включительно.

32-битовый цвет создаётся из четырёх 8-битовых каналов: альфа, красный, зелёный и синий. Эти каналы представлены как (A, R, G, B). При ссылке на источник, каналы обозначаются как A_s , R_s , G_s и B_s ; для получателя они A_d , R_d , G_d и B_d . Альфа-сопряжение примитивов драйвера поддерживает только базовое непрозрачное сопряжение с сопряжением источника:

$$(A_s, A_s, A_s, A_s) + ((1, 1, 1, 1) - (A_s, A_s, A_s, A_s)).$$

Альфа-сопряжение может быть использовано двумя способами:

- Как глобальный фактор, применяемый на каждый пиксель источника
или
- С картой, указывающей альфа-сопряжение для каждого отдельного пикселя. Альфа-карта "подцеплена" к началу Вашей команды прорисовки и является "черепичной", если размеры карты меньше, чем размеры операции прорисовки.

Функции включают:

<code>PgAlphaOff()</code>	Выключение операций альфа-сопряжения
<code>PgAlphaOn()</code>	Включение операций альфа-сопряжения
<code>PgAlphaValue()</code>	Извлечение альфа-компонента из значения цвета
<code>PgARGB()</code>	Преобразование значений альфа, красного, зелёного и синего в комбинированный формат цвета
<code>PgSetAlpha()</code>	Детальная установка параметров альфа-сопряжения
<code>PgSetAlphaBlend()</code>	Простая установка параметров альфа-сопряжения

Поддержка хроматического ключа

Операции хроматического ключа представляют собой метод маскирования пиксельных данных при операциях формирования изображения (копирования, формирования образа, прямоугольников, прочая), основанный на значении хроматического цвета. Базовыми режимами операции являются:

- Маскирование цвета ключа источника
- Маскирование цвета ключа получателя
- Маскирование всего, кроме цвета ключа источника
- Маскирование всего, кроме цвета ключа получателя

Функции включают:

PgChromaOff() Выключение операций хроматического ключа
 PgChromaOn() Включение операций хроматического ключа
 PgSetChroma() Установка хроматического ключа и операции

Операции расширенного растра

МикроGUI Photon'a поддерживает 256 растровых операций. Операции могут быть выполнены с использованием пиксельных данных источника, пиксельных данных получателя и пиксельных данных цветного расширения монохромного шаблона. Операции расширенного растра устанавливаются тем же образом, как и обычные растровые операции, используя функцию PgSetDrawMode().

Расширенные растровые операции являются распространяющимися, что означает, что они действуют на все последующие операции прорисовки, включая операции битирования бита (? bit-blit operations) и образы. Старый стиль растровых операций ещё существует, и его поведение такое же, как и в более поздних версиях микроGUI Photon'a.

Расширенные растровые операции определены как Pg_DrawModecharacters, в обратной записи, где characters выбирается из следующих символов:

Символ:	Означает:
P	Pattern – шаблон
S	Source – источник
D	Destination – получатель
o	OR – логическое ИЛИ
a	AND – логическое И
n	NOT – логическое отрицание
x	XOR – логическое исключающее ИЛИ

Например:

Pg_DrawModeS Копировать все данные источника
 Pg_DrawModePSo Логическое ИЛИ данных источника с данными шаблона

Полный список всех доступных растровых операций см. в <photon/Pg.h>.

Вот такой же код:

```
PdOffscreenContext_t *context1;  

    PhRect_t rsrc,rdst;
```

```

/* Инициализация внеэкранных областей и формирование данных по изображению,
   которое мы хотим в ней иметь */
...

/* Копирование образа, хранящегося во внеэкранных контекстах, на экран,
   применение операции OR к данным источника и шаблона вместе */
rsrc.ul.x = rdst.ul.x = rsrc.ul.y = rdst.ul.y = 0;
rsrc.lr.x = rdst.lr.x = rsrc.lr.y = rdst.lr.y = 100;

PgSetDrawMode (Pg_DrawModePSo);
PgSetFillDither (Pg_BLUE, Pg_BLACK, Pg_PAT_CHECKB8);
PgContextBlit (context1, &rsrc, NULL, &rdst);

/* Применение операции OR к шаблону в виде синей и чёрной шахматки -
   и к данным источника; копирование результата на экран */
PgFlush();

```

Видеорежимы

Видеорежим определяет, как выглядит экран (что Вы видите на мониторе). Описание режима включает:

Width	Ширина экрана в пикселях
Height	Высота экрана в пикселях
Pixel depth	"Глубина" пикселя – число бит представляющих пиксель. определяет, как много уникальных цветов Вы можете видеть на экране одновременно.
Refresh rate	Частота обновления кадров – число, указывающее, с какой частотой обновляется люминофор на ЭЛТ Вашего монитора (представлен в Hz).

Метод перечисления видеорежимов, применяемый в микроGUI Photon'a, схож со спецификацией VESA, где имеются "номера режимов" – численные представления ширины, высоты и пиксельной глубины видеорежима. Частота регенерации зависит от номера режима (это является отдельным членом в PgDisplaySettings_t). Номера режимов определяет драйвер, так что для одной видеокарты режим 640x480x8 может быть режимом 2, тогда как для другой – режимом 3022. Чтобы определить свойства какого-либо данного номера режима, используйте функцию PgGetVideoModeInfo(). Чтобы получить список номеров режимов, поддерживаемых конкретным графическим драйвером, используйте функцию PgGetVideoModeList().

Функциями для работы с видеорежимами являются:

PdSetTargetDevice()	Установить целевое устройство
PgGetGraphicsHWCaps()	Определить характеристики аппаратного обеспечения
PgGetVideoMode()	Получить текущий видеорежим
PgGetVideoModeInfo()	Получить информацию о видеорежиме
PgGetVideoModeList()	Запросить у графического драйвера список поддерживаемых им видеорежимов
PgSetVideoMode()	Установить текущий видеорежим

Вот некий простой код:

```

PgVideoModes_t ModeList;
PgVideoModeInfo_t ModeInfo;
PgDisplaySettings_t ModeSetting;
int I = 0, done = 0;

if (PgGetVideoModeList(&ModeList)) {
    /* Ошибка - драйвер этого не поддерживает */
}

/* Использовать для этого режима принятую по умолчанию частоту обновления */
ModeSetting.refresh = 0;

```

```

while (!done) {
    if (PgGetVideoModeInfo(ModeList.modes[i], &ModeInfo)) {
        /* Код ошибки */
    }

    if ((ModeInfo.width == 640) && (ModeInfo.height == 480) && (ModeInfo.bits_per_pixel == 16))
    {
        /* Мы нашли режим, который искали */
        done = 1;
        ModeSetting.mode = ModeList.modes[i];
    }

    i++;
    if (i >= ModeList.num_modes) {
        /* Ошибка - режим не найден */
        done=1;
    }
}

PgSetVideoMode (&ModeSetting);

```

Градиенты

Градиент – это постепенный переход одного цвета в другой. Библиотека Photon'a поддерживает:

- Градиенты уровня драйвера – быстрые, но не изощрённые. Точность жертвуется ради скорости.
- Градиенты уровня приложения – более медлительные, но более точные.

Градиенты уровня драйвера

Хотя библиотека Photon'a поддерживает большое многообразие градиентов (см. `PhImage_t`), бывают моменты, когда Вам понадобится простой градиент, формируемый без хранения его в `PhImage_t`. Поэтому к графическому драйверу добавлены несколько базовых операций формирования градиента:

`PgDrawGradient()` Потребовать от графического драйвера сформировать градиент

Градиенты уровня приложения

Эти функции позволяют ВАМ создать Ваши собственные градиенты:

<code>PgBevelBox()</code>	Рисовать прямоугольник с фасками, имеющий градиенты
<code>PgCalcColorContrast()</code>	Вычислить светлый и тёмный цвета для использования в градиенте
<code>PgContrastBevelBox()</code>	Рисовать прямоугольник с фасками, имеющий градиенты и заданный уровень контрастности
<code>PgDrawGradientBevelBox()</code>	Рисовать прямоугольник с фасками, имеющий градиенты и два плоских цвета

Видеверлей

Видеверлейная пересчётная схема (video overlay scaler) – это аппаратная возможность, позволяющая на прямоугольной области видимого экрана менять масштабированные версии отличающегося рисунка. Предварительно отмасштабированные видеокadres обычно размещены во внеэкранной памяти, и они выбираются из памяти и накладываются сверху на экранный образ рабочего стола в реальном времени с помощью видеверлейной пересчётной схемы.

Для контроля за тем, какая часть видеокadra видима, используется хромоключ. Обычно приложение отбирает цвет, являющийся цветом хроматического ключа, и рисует прямоугольник этого цвета там, где появляется видеосодержание. Когда другое окно приложения размещается сверху на приложении, воспроизводящем видео, окрашенный в хроматический цвет прямоугольник затемняется. Поскольку видеоаппаратное обеспечение запрограммировано на отображение видеосодержания экрана только тогда, когда рисуется цвет хромоключа, видео не демонстрируется, пока прямоугольник цвета хромоключа затемнён.

Следующие функции и типы данных имеют дело с видеоверлеем:

<code>PgConfigScalerChannel()</code>	Конфигурирование канала пересчётной схемы видеоверлея
<code>PgCreateVideoChannel()</code>	Создание канала для видеопотока
<code>PgDestroyVideoChannel()</code>	Освобождение ресурса, связанного с видеоканалом
<code>PgGetOverlayChromaColor()</code>	Возвращает цвет, используемый для оверлейных операций хромоключа
<code>PgGetScalerCapabilities()</code>	Получает характеристики пересчётной схемы видеоверлея
<code>PgNextVideoFrame()</code>	Получает индекс следующего видеобуфера для заполнения
<code>PgScalerCaps_t()</code>	Структура данных, описывающая характеристики пересчётной схемы видеоверлея
<code>PgScalerProps_t()</code>	Структура данных, описывающая свойства пересчётной схемы видеоверлея
<code>PgVideoChannel_t()</code>	Структура данных, описывающая канал видеоверлея

Пример

```
#include <stdio.h>

#include <Ph.h>

#define SRC_WIDTH 100
#define SRC_HEIGHT 100

#define DATA_FORMAT Pg_VIDEO_FORMAT_YV12

unsigned char *ybuf0, *ybuf1;
unsigned char *ubuf0, *ubuf1;
unsigned char *vbuf0, *vbuf1;

void grab_ptrs(PgVideoChannel_t *channel) {
    /* Буферы перемещались, получить указатели снова */
    ybuf0 = PdGetOffscreenContextPtr(channel->yplane1);
    ybuf1 = PdGetOffscreenContextPtr(channel->yplane2);
    ubuf0 = PdGetOffscreenContextPtr(channel->uplane1);
    ubuf1 = PdGetOffscreenContextPtr(channel->uplane2);
    vbuf0 = PdGetOffscreenContextPtr(channel->vplane1);
    vbuf1 = PdGetOffscreenContextPtr(channel->vplane2);

    if (channel->yplane1) fprintf(stderr, "ybuf0: %x, stride %d\n", ybuf0, channel->yplane1->pitch);
    if (channel->uplane1) fprintf(stderr, "ubuf0: %x, stride %d\n", ubuf0, channel->uplane1->pitch);
    if (channel->vplane1) fprintf(stderr, "vbuf0: %x, stride %d\n", vbuf0, channel->vplane1->pitch);

    if (channel->yplane2) fprintf(stderr, "ybuf1: %x, stride %d\n", ybuf1, channel->yplane2->pitch);
    if (channel->uplane2) fprintf(stderr, "ubuf1: %x, stride %d\n", ubuf1, channel->uplane2->pitch);
    if (channel->vplane2) fprintf(stderr, "vbuf1: %x, stride %d\n", vbuf1, channel->vplane2->pitch);
}

void overlay_example() {
    PgVideoChannel_t *channel;
    PgScalerCaps_t vcaps;
    PgScalerProps_t props;
    unsigned char *ptr;
    unsigned short *ptr16;
    int i = 0, j, k, index;
    int color;
    PhDrawContext_t *old;
    int rc;

    if ((channel = PgCreateVideoChannel(Pg_VIDEO_CHANNEL_SCALER, 0)) == NULL) {
        perror("PgCreateVideoChannel");
        exit(1);
    }
}
```

```

/* Проход по доступным форматам в поисках интересующего */
vcaps.size = sizeof (vcaps);
while (PgGetScalerCapabilities(channel, i++, &vcaps) == 0) {
    if (vcaps.format == DATA_FORMAT) break;
    vcaps.size = sizeof (vcaps);
}
if (vcaps.format != DATA_FORMAT) {
    fprintf(stderr, "Формат не поддерживается?\n");
    exit(1);
}

props.size = sizeof (props);
props.format = DATA_FORMAT;
props.viewport.ul.x = 20;
props.viewport.ul.y = 20;
props.viewport.lr.x = 600;
props.viewport.lr.y = 440;
props.src_dim.w = SRC_WIDTH;
props.src_dim.h = SRC_HEIGHT;
props.flags =
    Pg_SCALER_PROP_SCALER_ENABLE |
    Pg_SCALER_PROP_DOUBLE_BUFFER |
    Pg_SCALER_PROP_DISABLE_FILTERING;
if (PgConfigScalerChannel(channel, &props) == -1) {
    fprintf(stderr, "Не удалось сконфигурировать канал\n");
    exit(1);
}

grab_ptrs(channel);

for (i = 0; i < 100; i++) {
    index = PgNextVideoFrame(channel);
    delay(50);
    ptr = (void *) (index ? ybuf1 : ybuf0);
    color = rand() & 0xff;
    for (k = 0; k < props.src_dim.h; k++) {
        memset(ptr, color, channel->yplanel->pitch);
        ptr += channel->yplanel->pitch;
    }
}

props.flags &= ~Pg_SCALER_PROP_DISABLE_FILTERING;
switch (PgConfigScalerChannel(channel, &props)) {
    case -1:
        fprintf(stderr, " Не удалось сконфигурировать канал \n");
        exit(1);
        break;
    case 1:
        grab_ptrs(channel);
        break;
    case 0:
        default:
        break;
}

fprintf(stderr, "\"TV показ\" эффект\n");
for (i = 0; i < 1000; i++) {
    index = PgNextVideoFrame(channel);
    ptr = (void *) (index ? ybuf1 : ybuf0);
    for (k = 0; k < props.src_dim.h; k++) {
        for (j = 0; j < channel->yplanel->pitch; j++) *(ptr + j) = rand() & 0xff;
        ptr = (void *) ((char *)ptr + channel->yplanel->pitch);
    }
}

/* Установка хроматичности для монохромности в нейтральное */
ptr = ubuf0;
for (i = 0; i < props.src_dim.h; i++) {
    memset(ptr, 128, props.src_dim.w / 2);
    ptr += channel->yplanel->pitch;
}
ptr = vbuf0;
for (i = 0; i < props.src_dim.h; i++) {
    memset(ptr, 128, props.src_dim.w / 2);
    ptr += channel->vplanel->pitch;
}

if (rand() % 200 == 23) {
    props.viewport.ul.x = rand() % 400;
    props.viewport.ul.y = rand() % 300;
}

```

```

    props.viewport.lr.x = props.viewport.ul.x + SRC_WIDTH + rand() % 200;
    props.viewport.lr.y = props.viewport.ul.y + SRC_HEIGHT + rand() % 200;
    if (PgConfigScalerChannel(channel, &props) == 1) grab_ptrs(channel);
}
}

/*
 * В действительности это не нужно, поскольку видеоресурсы будут освобождены
 * автоматически при завершении приложения
 */
PgDestroyVideoChannel(channel);
}

int main(int argc, char *argv[]) {
    PhAttach(NULL, NULL);

    overlay_example();

    fprintf(stderr, "Нормальное завершение\n");
}

```

Слои

Некоторые графические адаптеры позволяют вам накладывать множество «экранов» на один. Каждый такой накладываемый экран называется «слоем».

Слои могут использоваться для совмещения независимых отображаемых элементов. Поскольку такое наложение осуществляется аппаратурой графического адаптера, оно может оказаться более эффективным, чем рисование всех необходимых элементов на одном слое. Так например, быстрый навигационный интерфейс может быть реализован как «прокручивающаяся» карта, отображаемая на одном слое – в «подложке», и всплывающие элементы графического интерфейса пользователя, такие как меню или Web-браузер – на другом слое.

Возможности отображения слоев отличаются в зависимости от графического контроллера и драйвера. Некоторые графические контроллеры не поддерживают слои. Различные слои на одном и том же дисплее могут иметь разные возможности. Следует использовать функцию `PgGetLayerCaps()` для того, чтобы определить, существует ли слой и каковы его возможности.

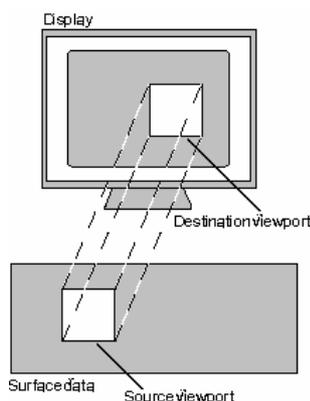
Слои нумеруются для каждого адаптера в отдельности начиная с нуля по возрастающей. Нумерация начинается со слоя, находящегося ниже всех остальных на данном адаптере. Слой может быть *активным* (отображается) или *неактивным* (не отображается). Может оказаться невозможно активировать слой, если его конфигурирование не выполнено полностью (если, например, не указан формат слоя, или для него выделено недостаточно *поверхностей*). Сконфигурировать слой можно только тогда, когда он является неактивным. После смены видеорежима конфигурация **всех** слоев на данном адаптере сбрасывается в установки по умолчанию.

Изображения на всех активных слоях адаптера объединяются используя механизмы альфа-сопряжения, хроматического ключа, или оба механизма одновременно – таким образом формируется окончательное изображение на экране.

Поверхности

Изображение, отображаемое на слое, берется с одного или нескольких внеэкранных контекстов, также называемых *поверхностями*. Количество поверхностей, необходимых для слоя, определяется его форматом. Так например, для слоя с форматом `Pg_LAYER_FORMAT_ARGB888` требуется только одна поверхность, тогда как для слоя с форматом `Pg_LAYER_FORMAT_YUV420` требуется три поверхности для получения полного изображения. Формат слоя устанавливается при помощи `PgSetLayerArg()`.

Окна просмотра



окно-источник и отображаемое окно

«Окно-источник» определяет прямоугольную область в данных поверхности. Окно используется для извлечения из данных поверхности того набора, который необходимо отобразить на слое.

«Отображаемое окно» определяет прямоугольную область на экране, где будет отображаться содержимое слоя.

Прокрутка и масштабирование, если поддерживаются слоем, могут осуществляться изменением этих типов «окон». Для перемещения по изображению на поверхности, связанной со слоем необходимо менять координаты «окна-источника». Для масштабирования – менять размеры «окон».



Вам необходимо установить цель действия этих преобразований используя `PdSetTargetDevice()`.

API слоев

Программный интерфейс слоев состоит из:

<code>PtGetLayerCaps()</code>	получить информацию о возможностях слоя
<code>PgCreateLayerSurface()</code>	создать внеэкранный поверхность, отображаемую слоем
<code>PgSetLayerSurface()</code>	отобразить внеэкранный поверхность на слое
<code>PgSetLayerArg()</code>	настроить параметры слоя
<code>PgLockLayer()</code>	захватить слой для эксклюзивного использования приложением
<code>PgUnlockLayer()</code>	снять захват со слоя
<code>PgLayerCaps_t</code>	структура данных, описывающая возможности слоя



API слоев в настоящий момент несовместимо с API оверлеев (`PgCreateVideoChannel()`, `PgConfigScalerChannel()`, `PgNextVideoFrame()`, и так далее). Не запускайте одновременно два приложения использующие эти два программные интерфейса.

Учтите:

- Photon не может рисовать во внеэкранных поверхностях, формат которых отличен от текущего видеорежима. Таким образом может оказаться невозможно использовать функции рисования Photon на внеэкранный онтексте, полученном при помощи `PgCreateLayerSurface()`. Вместо этого приложение должно использовать

PdGetOffscreenContextPtr(), для получения указателя на видеопамять и записывать данные напрямую.

- Если приложение изменит внеэкранный поверхность основного экрана при помощи PgSetLayerSurface(), Photon все равно продолжит рисовать в старую внеэкранный поверхность. Приложению необходимо сохранить указатель на старую поверхность и восстановить её как только оно перестанет использовать слой. Как это сделать показано ниже в примере.

Использование слоев

Чтобы использовать возможности слоев вы обычно должны сделать следующее:

1. Используйте PgGetLayerCaps() с постоянно увеличивающимся индексом слоя для определения возможностей оборудования (если вы еще их не знаете). Если PgGetLayerCaps() дает ошибку для любых значений – драйвер не поддерживает слои.
2. Если вы хотите предотвратить использование слоя другими приложениями – используйте PgLockLayer().
3. Создайте поверхности для отображения на слое и внеэкранные контексты – для отображения данных на поверхность при помощи PgCreateLayerSurface().
4. Вызовите PgSetLayerArg() с аргументом Pg_LAYER_ARG_LIST_BEGIN.
5. Вызовите PgSetLayerArg() для установки параметров слоя.
6. Вызовите PgSetLayerSurface() для отображения внеэкранный контекста поверхности на слое.
7. Вызовите PgSetLayerArg() для установки параметров слоя. Вы можете использовать Pg_LAYER_ARG_ACTIVE в качестве аргумента для отображения содержимого слоя на экране.
8. Вызовите PgSetLayerArg() с аргументом Pg_LAYER_ARG_LIST_END.
9. Если формат слоя RGB или PAL8 установите один из контекстов поверхности слоя текущим а затем используйте Pg* - функции для рисования во внеэкранный контекст.
10. Если формат слоя YUV, или подобный – вы чаще всего будете записывать данные непосредственно в буфер (например для воспроизведения видеопотока).
11. Если вы захватили слой для эксклюзивного использования вам необходимо снять блокировку при помощи PgUnlockLayer() прежде, чем ваше приложение выйдет.

Программа, приведенная ниже показывает использование программного интерфейса слоев.

Пример

```
#include <errno.h>
#include <stdio.h>
#include <Ph.h>

int
FindFormatIndex(int layer, unsigned int format)
{
    PGLayerCaps_t caps;
    int format_idx = 0;

    while (PgGetLayerCaps(layer, format_idx, &caps) != -1) {
        if (caps.format == format)
            return format_idx;

        format_idx++;
    }
    return -1;
}

int
main(int argc, char **argv)
{
    /*
     * For best results, these values should match your video mode.
     */
#define LAYER_FORMAT      Pg_LAYER_FORMAT_ARGB8888
#define SURFACE_WIDTH    1024
#define SURFACE_HEIGHT   768
```

```

struct _Ph_ctrl          *ph;
PgLayerCaps_t           caps;
PdOffscreenContext_t    *surf;
PdOffscreenContext_t    *scr = NULL;
PhDrawContext_t         *olddc;
PhRid_t                 driver_rid = -1;
int    layer_idx = -1;
int    format_idx = -1;
int    active = 1;
int    i;

/*
 * Arguments:
 * -d <driver region>
 * -l <layer index>
 */

while ((i = getopt(argc, argv, "d:l:")) != -1) {
    switch(i) {
        case 'd': /* driver region */
            driver_rid = atol(optarg);
            break;
        case 'l': /* layer index */
            layer_idx = atoi(optarg);
            break;
        default:
            break;
    }
}

if (layer_idx == -1) {
    printf("Specify layer index.\n");
    exit(-1);
}

if (driver_rid == -1) {
    printf("Specify graphics driver region.\n");
    exit(-1);
}

ph = PhAttach(NULL, NULL);
if (ph == NULL) {
    perror("PhAttach");
    exit(-1);
}

if (-1 == PdSetTargetDevice(PhDCGetCurrent(), driver_rid)) {
    perror("PdSetTargetDevice");
    exit(-1);
}

/* Check if the layer supports the required format */
format_idx = FindFormatIndex(layer_idx, LAYER_FORMAT);
if (format_idx == -1) {
    printf("Layer does not support format\n");
    exit(-1);
}

/* Get the layer capabilities */
PgGetLayerCaps(layer_idx, format_idx, &caps);

if (caps.caps & Pg_LAYER_CAP_MAIN_DISPLAY) {
    /* Save a reference to the current display surface */
    scr = PdCreateOffscreenContext(0, 0, 0, Pg_OSC_MAIN_DISPLAY);
}

/* Allocate a surface for the layer */
surf = PgCreateLayerSurface(layer_idx, 0, format_idx,
    SURFACE_WIDTH, SURFACE_HEIGHT, Pg_OSC_MEM_PAGE_ALIGN);
if (surf == NULL)
    exit(-1);

/* Draw some stuff on the surface */
olddc = PhDCSetCurrent(surf);
PgSetFillColor(Pg_BLACK);
PgDrawIRect(0, 0, SURFACE_WIDTH-1, SURFACE_HEIGHT-1, Pg_DRAW_FILL);
PgSetFillColor(Pg_YELLOW);
PgDrawIRect(0, 0, 100, 100, Pg_DRAW_FILL);
PgSetFillColor(PgRGB(255,180, 0));

```

```
PgDrawIRect(70, 80, 600, 500, Pg_DRAW_FILL);
PhDCSetCurrent(olddc);

/* Lock the layer */
if (-1 == PgLockLayer(layer_idx))
    exit(-1);

/* Start configuring arguments */
PgSetLayerArg(layer_idx, Pg_LAYER_ARG_LIST_BEGIN, 0, 0);

/* Select the layer format */
PgSetLayerArg(layer_idx, Pg_LAYER_ARG_FORMAT_INDEX,
    &format_idx, sizeof(int));

/* This changes the current display surface */
PgSetLayerSurface(layer_idx, 0, surf);

PgSetLayerArg(layer_idx, Pg_LAYER_ARG_ACTIVE,
    &active, sizeof(int));

/* Configure other arguments ... */

/* End configuration */
PgSetLayerArg(layer_idx, Pg_LAYER_ARG_LIST_END, 0, 0);

/* Application continues ... */
sleep(3);

/* Finished using layer; Restore the current display surface */
if (caps.caps & Pg_LAYER_CAP_MAIN_DISPLAY) {
    PgSetLayerArg(layer_idx, Pg_LAYER_ARG_LIST_BEGIN, 0, 0);
    PgSetLayerSurface(layer_idx, 0, scr);
    PgSetLayerArg(layer_idx, Pg_LAYER_ARG_LIST_END, 0, 0);
}

PgUnlockLayer(layer_idx);

if (scr) PhDCRelease(scr);
PhDCRelease(surf);

PhDetach(ph);
exit(0);
}
```

Глава 19. Шрифты

Хотя библиотеки Photon'a и предлагают ряд функций по работе с шрифтами (см. главу "Pf – сервер шрифта" в "Справочнике библиотечных функций Photon'a"), большинство из них являются подпрограммами низкого уровня и, вероятно, Вам не понадобится их использовать. Эта глава описывает основы использования шрифтов.

Эта глава включает:

- Метрики символа
- Имена шрифтов
- Написание текста в прямоугольной области
- Пропорциональный текст, приводящий к ошибкам восстановления повреждений.

Метрики символа

Давайте начнём с некоторых определений:

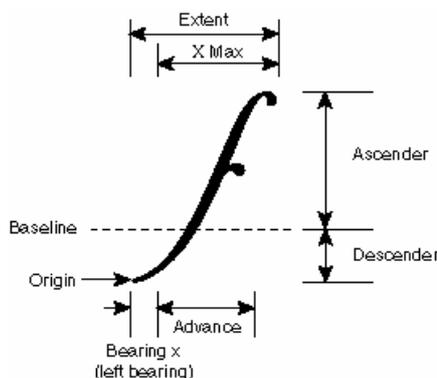


Рис. 19-1. Метрики символа

Advance	Продвижение. Величина, на которую продвигается по оси x перо после прорисовки литеры. Это может быть не полная ширина литеры (особенно для курсивного шрифта) для обеспечения кернинга (т.е. создания выносного элемента литеры).
Ascender	Верхний элемент литеры. Высота от базовой линии до вершины литеры.
Bearing x or left bearing	Выноска по x или левая выноска. Размер литеры слева оттуда, где символ считается начинающимся.
Descender	Подстрочный элемент литеры. Высота от низа литеры до базовой линии.
Extent	Протяжённость литеры. Ширина литеры. Зависит от шрифта, она может включать определённое чистое пространство.
Origin	Начало. Нижний левый угол литеры.
X Max	Ширина символа, не включая выноски по x.

☞ Для экономии времени и памяти, кернинг не поддерживается.

Имена шрифтов

Шрифт идентифицируется по своему имени, которое может иметь одну из следующих форм:

Имя лигатуры (foundry name)	Имя, задаваемое именем лигатуры для идентификации семейства шрифтов, такие как Helvetica, Comic Sans MS или Prima Sans BT. Обратите внимание на использование заглавных букв. Имя лигатуры не включает информацию о стиле (напр., жирный, наклонный) или размере. Это имя является универсальным для всего операционного окружения (напр., X, Photon).
Имя основы	Уникальный идентификатор, включающий сокращение от имени лигатуры, а также стиль (напр., b, i) и размер. Например, helv12 является именем основы 12-пунктного шрифта Helvetica, а helv12b – имя основы 12-пунктного жирного шрифта Helvetica.

Чтобы задать шрифт в API Photon'a, обычно используется имя основы. Вы должны рассматривать имена основы как постоянные идентификаторы, а не как модифицируемые строки.

Вы можете жёстко прописать все ссылки на шрифты в приложении Photon'a. Но Ваше приложение может быть более гибким, если Вы используете лигатурное имя, чтобы иметь возможность выбора наилучшего совпадения из всех доступных шрифтов. При таком подходе нет проблемы, если какой-то конкретный шрифт со временем окажется переименован, удалён или перемещён. Например, в следующем вызове функции PtAlert() используется жёстко прописанное имя основы helv14, задающее 14-точечный шрифт Helvetica:

```
answer = PtAlert(base_wgt, NULL, "File Not Saved", NULL,
                "File has not been saved.\nSave it?",
                "helv14", 3, btns, NULL, 1, 3, Pt_MODAL );
```

Доступные имена основы Вы можете получить из имён файлов в `#{PHOTON_PATH}/font` – просто удалите имеющееся расширение файла (напр., .phf).

Иной способ заключается в том, что если у Вас есть директория `$HOME/.ph`, просмотреть директорию `$HOME/.ph/font/`. МикроGUI Photon'a создаёт этот локальный файл только при необходимости, как скажем, когда Вы запускаете утилиту *fontadmin* (см. "Справочник утилит QNX 6") для создания Вашей собственной персональной конфигурации. Пока локальный файл не создан, микроGUI использует глобальный файл.

Запрос доступных шрифтов

В приведенном выше примере используется сокращённое наименование жёстко прописанного имени основы (helv14). И, как любое приближение, оно допускает компромиссы. Прежде всего имена основы являются объектами для изменений. Что более важно, все версии Photon'овского микроGUI, вплоть до версии 1.13 включительно, позволяют использовать для имени основы не более 16 символов. Этого не всегда достаточно, чтобы дать каждому шрифту уникальную основу. Текущая версия микроGUI Photon'a допускает до 80 символов.

Чтобы обойти эту проблему, Вы можете использовать функцию `PfQueryFonts()`, чтобы определить, какие шрифты доступны, и обеспечить информацию, требующуюся для построения имени основы. Эта функция запрашивает сервер шрифтов Photon'a и защищает Вас от будущих изменений. Давайте начнём с параметров функции `PfQueryFonts()` – а затем рассмотрим образец программного кода, в котором из данных, возвращаемых функцией, извлекается имя основы. Сама функция выглядит подобным образом:

```
PfQueryFonts (long symbol,
              unsigned flags,
              FontDetails list[],
              int n );
```

Её аргументами являются:

- symbol* Ключ поиска для Photon'овского менеджера шрифтов. Функция ищет шрифты, которые включают этот символ, и отбрасывает те, которые его не имеют. Например, символ пробела в Unicode (0x0020) доступен почти во всех шрифтах. С другой стороны, задание символа "ё" (в Unicode код символа 0x00C9) существенно сужает выбор шрифтов. И конечно, задание японского символа приведёт к отбору только японских шрифтов. Список символов см. в PkKeyDef.h или в ISO/EIC 10646-1. Чтобы включить все доступные шрифты, используйте константу PHFONT_ALL_SYMBOLS.
- flags* Предлагает другой способ сужения круга поиска. Возможными значениями этого параметра являются:
- PHFONT_SCALABLE – эти шрифты используют наборы векторов для описания каждого символа, делая возможным отображение шрифта различного размера.
 - PHFONT_BITMAP – эти шрифты хранят в качестве своих символов настоящие картинки.
 - PHFONT_PROP – эти шрифты являются пропорциональными. В пропорциональных шрифтах символ, например, "w" шире, чем символ "i".
 - PHFONT_FIXED – эти шрифты делают все символы одной ширины.
 - PHFONT_ALL_FONTS [Почему-то ничего не написано, но полагаю, означает включение всех доступных шрифтов – Прим. пер.]
 - PHFONT_DONT_SHOW_LEGACY – исключаются унаследованные шрифты из более ранних версий микроGUI Photon'a. Этот флаг перекрывает флаг PHFONT_ALL_FONTS.
- list[]* Массив, который Photon'овский менеджер шрифтов заполняет для Вас. Вы должны объявить структуру FontDetails, описанную ниже.
- n* Число элементов, доступных в списочном массиве.

Если функция PfQueryFonts() завершилась успешно, она возвращает число доступных шрифтов, совпавших с заданным критерием отбора. В противном случае она возвращает -1.

☞ Если *n* равно 0 и *list* равен NULL, функция PfQueryFonts() возвращает число совпавших шрифтов, но не пытается заполнять список. Вы можете использовать эту возможность для определения количества элементов при выделении памяти под список.

Структура FontDetails

После получения списка шрифтов Вы должны проверить в нём структуру FontDetails, чтобы найти нужный Вам шрифт и определить строку для использования её в качестве имени основы. Структура FontDetails определена в <photon/Pf.h> следующим образом:

```
typedef struct {
    FontDescription desc;
    FontName        stem;
    short           losize;
    short           hisize;
    unsigned short flags;
} FontDetails;
```

Для наших целей наиболее интересны элементы *desc* и *stem*, но давайте рассмотрим их все:

- desc* Имя лигатуры или полное описательное имя шрифта, такое как "Helvetica" или "Charter".
- stem* Краткая форма. Она предоставляет часть имени основы, используемую вызовами API Photon'a. Например, "helv" и "char" соответствуют "Helvetica" и "Charter".
- losize* Минимальный возможный размер шрифта в пунктах, скажем 4.
- hisize* Наибольший возможный размер шрифта. Если и *losize* и *hisize* равны 0, то шрифт масштабируемый.
- flags* Возможные значения:
- PHFONT_INFO_FIXED – шрифт постоянной ширины
 - PHFONT_INFO_PROP – пропорциональный шрифт

- PHFONT_INFO_PLAIN – шрифт не является ни жирным, ни курсивным
- PHFONT_INFO_BOLD – жирный шрифт
- PHFONT_INFO_ITALIC – курсивный шрифт

Генерирование имён шрифтов

Как описано выше, Photon'овский API требует для идентификации шрифта имя основы, но если Вы хотите быть гибким, можете использовать лигатурное имя шрифта.

Простейшим способом получения имени основы, задав лигатурное имя шрифта, желаемый размер в пунктах и стиль, является вызовом функции PfGenerateFontName(). Она создаёт в предоставляемом Вами буфере уникальное имя основы для шрифта. (Вы можете использовать этот подход, даже если Вы не используете функцию PfQueryFonts() для отыскания всех доступных шрифтов). Прототип этой функции следующий:

```
char * PfGenerateFontName(
    char const * pkcDescription,
    uint32_t    kuiFlags,
    uint32_t    kuiSize,
    char        * pcBuff );
```

Если функция PfGenerateFontName() завершилась успешно, она возвращает указатель на буфер; в случае неудачи возвращается NULL.

Мы определили для Вас тип данных FontName для использования в буфере, переданном функции PfGenerateFontName(). Это массив размером MAX_FONT_TAG. Для успешного программирования шрифта не используйте буфер хранения идентификатора шрифта размером меньшим, чем FontName. Вот вызов функции PtAlert() – такой же, как показан выше, но на этот раз используется вызов PfGenerateFontName():

```
char Helvetica14[MAX_FONT_TAG];

if ( PfGenerateFontName("Helvetica", 0, 14, Helvetica14) == NULL ) {
    /* Не удалось найти шрифт! */
    ...
}

answer = PtAlert( base_wgt, NULL, "File Not Saved", NULL, "File has not been saved.\nSave it?",
    Helvetica14, 3, btns, NULL, 1, 3, Pt_MODAL );
```

Теперь то, что мы рассматривали по кусочкам, просто чётко следует по пунктам, необходимым для сборки корректного имени основы для данного шрифта.

Имейте в виду следующие соображения:

- Используйте буфер FontName для размещения в нём имени основы.
- Поиск шрифта основан на имени лигатуры (т.е. члена desc его входа FontDetails), а не на имени основы stem.

Вы, вероятно, захотите выполнить эту работу в инициализирующей функции Вашего приложения, или, возможно, в установочной функции базового окна. Определите буфер FontName как глобальную переменную; Вы сможете затем использовать это имя в Вашем приложении по мере необходимости.

Вот простая функция инициализации приложения:

```
/******
*** Глобальные переменные ***
*****/

FontName GcaCharter14Bold;

int fcnAppInit( int argc, char *argv[] ) {
    /* Локальные переменные */
    FontDetails tsFontList [nFONTLIST_SIZE];
    short sCurrFont = 0;
    char caBuff[20];
```

```

/* Получение описания доступных шрифтов */
if (PfQueryFonts (PHFONT_ALL_SYMBOLS, PHFONT_ALL_FONTS,
                 tsFontList, nFONTLIST_SIZE) == -1) {
    perror ("PfQueryFonts() неудача: ");
    return (Pt_CONTINUE);
}

/* Поиск среди них шрифта, совпадающего с нашими спецификациями */
for (sCurrFont = 0; sCurrFont < nFONTLIST_SIZE; sCurrFont++) {
    if ( !strcmp (tsFontList[sCurrFont].desc, "Charter") ) break; /* мы нашли его */
}

/* Переполнение проверки */
if (sCurrFont == nFONTLIST_SIZE) {
    /* проверка на частичное совпадение */
    for (sCurrFont = 0; sCurrFont < nFONTLIST_SIZE; sCurrFont++) {
        if ( !strncmp (tsFontList[sCurrFont].desc, "Charter", strlen ("Charter")))
            break; /* найдено частичное совпадение */
    }

    if (sCurrFont == nFONTLIST_SIZE) {
        printf ("шрифта Charter нет в %d проверенных шрифтах.\n", sCurrFont);
        return (Pt_CONTINUE);
    }
    else printf ("Используется частичное совпадение -- 'Charter'.\n");
}

/* Имеет ли он жирный вариант? */
if (!(tsFontList[sCurrFont].flags & PHFONT_INFO_BOLD)) {
    printf ("Charter не доступен как жирный.\n");
    return (Pt_CONTINUE);
}

/* Доступен ли 14-пунктный? */
if ( !( (tsFontList[sCurrFont].losize == tsFontList[sCurrFont].hisize == 0)
/* пропорциональный шрифт - а он может быть изображён в 14 пунктов */

    ||

    ( (tsFontList[sCurrFont].losize <= 14 )
      &&
      (tsFontList[sCurrFont].hisize >= 14 ) ) ) )
/* 14-пунктный уместается между наименьшим и наибольшим доступными размерами */

{
    printf ("шрифт Charter не доступен как 14-пунктный.\n");
    return (Pt_CONTINUE);
}

/* Генерирование имени основы */
if (PfGenerateFontName( tsFontList[sCurrFont].desc,
                       PF_STYLE_BOLD, 14, GcaCharter14Bol) == NULL) {
    perror ("PfGenerateFontName() неудача: ");
    return (Pt_CONTINUE);
}

/* Теперь Вы можете использовать GcaCharter14Bold как аргумент в PtAlert(), etc. */

/* предотвращает предупреждения (варнинги) об отсутствии ссылок */
argc = argc, argv = argv;

return( Pt_CONTINUE );
} // функции fcnAppInit()
Чтобы вышеприведенный код работал, Вы должны объявить в глобальном хедер-файле приложения
нижеследующую информацию. Чтобы это сделать, используйте диалог PhAB'a "Startup Info/Modules"
(доступный из меню "Application").

/*****
*** определённые пользователем константы ***
*****/
#define nFONTLIST_SIZE 100 /* просто случайно выбранный размер */

/*****
*** глобальные переменные ***
*****/

```

```
extern FontName GcaCharter14Bold;
```

Вы можете избежать использования заданного размера списка, вызвав функцию `PfQueryFonts()` с параметром *n*, установленным в 0, и параметром *list* – в NULL. Если Вы так сделаете, функция `PfQueryFonts()` вернёт количество совпавших шрифтов, но не будет пытаться заполнить список. Вы можете использовать эту возможность для определения числа записей при выделении памяти.

Помните о том, что определить этот хедер-файл надо перед тем, как Вы начнёте добавлять ответные реакции и установочные функции – в этом случае это будет автоматически включено как `#define`. Если Вы забудете сделать это, Вам придётся вернуться назад и добавить оператор ручками. Более полно см. раздел "Задание глобального хедер-файла" в главе "Работа с приложениями".

И, наконец, вот пример ответной реакции, которая использует нашу строку с именем основы:

```
int fcnbase_btn_showdlg_ActivateCB( PtWidget_t *widget,
                                   ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo ) {

/* Эта ответная реакция используется для запуска диалогового окна
   с целью поупражняться с глобальной переменной GcaCharter14Bold */

PtNotice (ABW_base, NULL, "Демонстрация шрифта ", NULL,
          "Это написано 14-пунктовым жирным шрифтом Charter",
          GcaCharter14Bold, "OK", NULL, 0);

/* предотвращает предупреждения (варнинги) об отсутствии ссылок */
widget = widget, apinfo = apinfo, cbinfo = cbinfo;

return( Pt_CONTINUE );
}
```

Написание текста в прямоугольной области

Написание текста в прямоугольнике заданного размера может оказаться непростым делом, если неизвестен размер строки. Рассмотрим прямоугольник фиксированных размеров, например, ячейку электронной таблицы. Как вы определите, сколько символов можно успешно отобразить в этой ячейке без отсечения? Вызовите функцию `PfExtentTextToRect()`. Передайте ей отсекающий прямоугольник, идентификатор шрифта, строку, максимальное число байтов в строке, и она сообщит Вам число символов и занимаемое ими пространство, которые уменьшаются внутри отсекающего прямоугольника. Это полезно для размещения многоточий (...) после обрезанной строки и недопущения частично обрезанных символов. В настоящее время эта функция поддерживает отсечение только по горизонтальной оси.

Вот пример:

```
/* PfExtentTextToRect */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <Ap.h>
#include <Ph.h>
#include <Pt.h>
#include <errno.h>

PtWidget_t * pwndMain = NULL, * pbtn = NULL, * pObjRaw = NULL;
char * pcText = "pAfaBfbfffffffffffffffffCfcXfxYfyZfzf";
char * pcGB = "\323\316\317\267";
char ** ppcData = NULL;

int fnDrawCanvas( PtWidget_t * ptsWidget, PhTile_t * ptsDamage );

#define FALSE 0

FontName szFont;

char * pmbGB = NULL;
struct P×TransCtrl * ptsTrans = NULL;
```

```

int iTemp1 = 0, iTemp2 = 0;

#define BUFFER_SIZE 256

int main (int argc, char *argv[]) {
    PtArg_t args[4];
    PhPoint_t win_size, pntPOS, pntDIM;
    short nArgs = 0;

    if ((pmbGB = calloc(BUFFER_SIZE, sizeof(char))) == NULL) return(EXIT_FAILURE);

    PtInit (NULL);

    if (argc > 1) {
        if (PfGenerateFontName(argv[1], 0, 9, szFont) == NULL)
            PfGenerateFontName("TextFont", 0, 9, szFont);
    }
    else PfGenerateFontName("TextFont", 0, 9, szFont);

    if ((ptsTrans = PxTranslateSet (NULL, "GB2312-80")) == NULL) return(EXIT_FAILURE);

    if (PxTranslateToUTF(ptsTrans, pcGB, 4, &iTemp1, pmbGB, BUFFER_SIZE, &iTemp2) == -1)
        printf("Не получается перевести из GB в UTF.\n");

    if(argc > 2) pcText = pmbGB;

    /* Установка базовых параметров pwndMain */
    win_size.x = 450;
    win_size.y = 450;

    PtSetArg(&args[0], Pt_ARG_DIM, &win_size, 0);
    PtSetArg(&args[1], Pt_ARG_WINDOW_TITLE, "PfExtentTextToRect", 0);

    pwndMain = PtCreateWidget (PtWindow, Pt_NO_PARENT, 2, args);

    nArgs = 0;
    pntPOS.x = 100;
    pntPOS.y = 10;
    PtSetArg(&args[nArgs], Pt_ARG_POS, &pntPOS, 0);
    nArgs++;
    PtSetArg(&args[nArgs], Pt_ARG_TEXT_STRING, pcText, NULL);
    nArgs++;
    PtSetArg(&args[nArgs], Pt_ARG_TEXT_FONT, szFont, NULL);
    nArgs++;
    pbtn = PtCreateWidget (PtButton, pwndMain, nArgs, args);
    PtRealizeWidget (pbtn);

    pntPOS.y = 100;
    pntPOS.x = 75;
    pntDIM.x = 300;
    pntDIM.y = 300;
    PtSetArg(&args[0], Pt_ARG_POS, &pntPOS, 0);
    PtSetArg(&args[1], Pt_ARG_DIM, &pntDIM, 0);
    PtSetArg(&args[2], Pt_ARG_RAW_DRAW_F, fnDrawCanvas, 0L);
    pobjRaw = PtCreateWidget (PtRaw, pwndMain, 3, args);

    PtRealizeWidget (pwndMain);

    PtMainLoop ();

    return(0);
} // main()

#define ASCENDER tsExtent.ul.y
#define DESCENDER tsExtent.lr.y

int fnDrawCanvas( PtWidget_t * ptsWidget, PhTile_t * ptsDamage ) {
    PhRect_t tsExtentClip;
    PhRect_t rect;
    PhPoint_t pnt;
    PhRect_t tsExtent;
    PgColor_t old;
    PhPoint_t pnt2;
    PhPoint_t tsPos = {0, 0};
    int iRet = 0;
    int iBytes = 0;

    /* Находим наш холст */
    PtBasicWidgetCanvas (pobjRaw, &rect);

```

```
old = PgSetStrokeColor(Pg_BLACK);

PfExtentText(&tsExtent, &tsPos, szFont, pcText, strlen(pcText));

/* Рисуем текст */
pnt.x = 10 + rect.ul.x;
pnt.y = 100 + rect.ul.y;

PgSetFont(szFont);
PgSetTextColor(Pg_BLACK);
PgDrawText(pcText, strlen(pcText), &pnt, 0);

pnt.x -= 10;
pnt2.x = pnt.x + tsExtent.lr.x + 20;
pnt2.y = pnt.y;

PgSetStrokeColor(Pg_BLUE);

PgDrawLine(&pnt, &pnt2);

pnt.x = 10 + rect.ul.x;
pnt.y = 100 + rect.ul.y;

PgSetStrokeColor(Pg_RED);

PgDrawIRect(tsExtent.ul.x + pnt.x, tsExtent.ul.y + pnt.y,
            (tsExtent.lr.x - min(tsExtent.ul.x, 0) + 1) + pnt.x,
            tsExtent.lr.y + pnt.y,
            Pg_DRAW_STROKE);

if ((iRet = PfExtentTextToRect(&tsExtentClip, szFont, &tsExtent, pcText, strlen(pcText))) == -1)
printf("PfExtentTextToRect неудача 1.\n");
else {
    printf("lrx == %d, %d символов в строке.\n", tsExtent.lr.x, utf8strlen(pcText, &iBytes));
    printf("PfExtentTextToRect lrx == %d, %d символов разместятся в обрезке %d.\n",
           tsExtentClip.lr.x, iRet, tsExtent.lr.x);
}

tsExtent.lr.x /= 2;

if ((iRet = PfExtentTextToRect(&tsExtentClip, szFont, &tsExtent, pcText, strlen(pcText))) == -1)
printf("PfExtentTextToRect неудача 2.\n");
else {
    printf("lrx == %d, %d символов в строке.\n", tsExtent.lr.x, utf8strlen(pcText, &iBytes));
    printf("PfExtentTextToRect lrx == %d, %d символов разместятся в обрезке %d.\n",
           tsExtentClip.lr.x, iRet, tsExtent.lr.x);
}

pnt.x = 10 + rect.ul.x;
pnt.y = 150 + rect.ul.y;

PgDrawText(pcText, iRet, &pnt, 0);
PgDrawIRect(tsExtentClip.ul.x + pnt.x,
            tsExtentClip.ul.y + pnt.y,
            (tsExtentClip.lr.x - min(tsExtentClip.ul.x, 0) + 1) + pnt.x,
            tsExtentClip.lr.y + pnt.y,
            Pg_DRAW_STROKE);

tsExtent.lr.x /= 2;

if ((iRet = PfExtentTextToRect(&tsExtentClip, szFont, &tsExtent, pcText, strlen(pcText))) == -1)
printf("PfExtentTextToRect неудача 3.\n");
else {
    printf("lrx == %d, %d символов в строке.\n", tsExtent.lr.x, utf8strlen(pcText, &iBytes));
    printf("PfExtentTextToRect lrx == %d, %d символов разместятся в обрезке %d.\n",
           tsExtentClip.lr.x, iRet, tsExtent.lr.x);
}

pnt.x = 10 + rect.ul.x;
pnt.y = 200 + rect.ul.y;

PgDrawText(pcText, iRet, &pnt, 0);
PgDrawIRect(tsExtentClip.ul.x + pnt.x, tsExtentClip.ul.y + pnt.y,
            (tsExtentClip.lr.x - min(tsExtentClip.ul.x, 0) + 1) + pnt.x,
            tsExtentClip.lr.y + pnt.y,
            Pg_DRAW_STROKE);

PgSetStrokeColor(old);
return( Pt_CONTINUE );
}
```

Исправление повреждений в случае пропорционального шрифта текста

При действии с пропорциональными шрифтами, иногда векторы одного глифа попадают на векторы другого. Это особенно видно, когда используются такие шрифты, как Nuptial BT. Чтобы исправлять повреждения в случае таких шрифтов, необходимо применять особые меры.

Для этого предназначена функция `PfExtentTextCharPositions()`. Вы можете использовать её, чтобы получить позицию после каждого символа, включая выноску по *x* следующего символа. Эта позиция – та, где Вы должны рисовать следующий символ. Если Вы используете флаг `PF_CHAR_DRAW_POSITIONS`, выноска по *x* следующего символа не добавляется, что полезно, когда Вы позиционируете курсор.

Например:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <Ap.h>
#include <Ph.h>
#include <Pt.h>
#include <errno.h>

PtWidget_t * pwndMain = NULL,
            * pbtn = NULL,
            * pObjRaw = NULL,
            * pObjLabel = NULL;
char ** ppcData = NULL;

int fnDrawCanvas( PtWidget_t * ptsWidget, PhTile_t * ptsDamage );

#define FALSE 0

#define __WIN_SIZE_X_ 1000

FontName szFont;

int main (int argc, char *argv[]) {
    PtArg_t args[8];
    PhPoint_t win_size, pntPOS, pntDIM;
    short nArgs = 0;
    char caTitle[50];

    if (argc < 2) {
        printf("Usage: pen text_string\n");
        exit(EXIT_FAILURE);
    }

    PtInit (NULL);

    ppcData = argv;

    PfGenerateFontName("TextFont", 0, 9, szFont);

    /* Установка базовых параметров pwndMain */
    win_size.x = 800;
    win_size.y = 600;

    sprintf(caTitle, "Получение позиции пера");
    PtSetArg(&args[0], Pt_ARG_DIM, &win_size, 0);
    PtSetArg(&args[1], Pt_ARG_WINDOW_TITLE, caTitle, 0);

    pwndMain = PtCreateWidget (PtWindow, Pt_NO_PARENT, 2, args);

    nArgs = 0;
    pntDIM.x = 80;
    pntDIM.y = 20;
    PtSetArg(&args[nArgs], Pt_ARG_DIM, &pntDIM, 0);
    nArgs++;
    pntPOS.x = 100;
    pntPOS.y = 10;
    PtSetArg(&args[nArgs], Pt_ARG_POS, &pntPOS, 0);
    nArgs++;
    PtSetArg(&args[nArgs], Pt_ARG_TEXT_STRING, argv[1], NULL);
    nArgs++;
    pbtn = PtCreateWidget(PtButton, pwndMain, nArgs, args);
```

```

PtRealizeWidget(pbtn);

nArgs = 0;
pntDIM.x = 80;
pntDIM.y = 20;
PtSetArg(&args[nArgs], Pt_ARG_DIM, &pntDIM, 0);
nArgs++;
pntPOS.x = 100;
pntPOS.y = 600;
PtSetArg(&args[nArgs], Pt_ARG_POS, &pntPOS, 0);
nArgs++;
PtSetArg(&args[nArgs], Pt_ARG_TEXT_STRING, argv[1], NULL);
nArgs++;
PtSetArg(&args[nArgs], Pt_ARG_RESIZE_FLAGS,
         Pt_RESIZE_XY_ALWAYS, Pt_RESIZE_XY_ALWAYS);
nArgs++;
PtSetArg(&args[nArgs], Pt_ARG_BORDER_WIDTH, 0L, 0L);
nArgs++;
PtSetArg(&args[nArgs], Pt_ARG_MARGIN_LEFT, 0L, 0L);
nArgs++;
PtSetArg(&args[nArgs], Pt_ARG_MARGIN_RIGHT, 0L, 0L);
nArgs++;
pobjLabel = PtCreateWidget(PtLabel, pwndMain, nArgs, args);
PtRealizeWidget(pobjLabel);

pntPOS.y = 100;
pntPOS.x = 75;
pntDIM.x = _WIN_SIZE_X_ - 75 - 10;
pntDIM.y = 300;
PtSetArg(&args[0], Pt_ARG_POS, &pntPOS, 0);
PtSetArg(&args[1], Pt_ARG_DIM, &pntDIM, 0);
PtSetArg(&args[2], Pt_ARG_RAW_DRAW_F, fnDrawCanvas, 0L);
pobjRaw = PtCreateWidget(PtRaw, pwndMain, 3, args);

(void) PtRealizeWidget(pwndMain);

PtMainLoop ();

return(0);
} // main()

int fnDrawCanvas( PtWidget_t * ptsWidget, PhTile_t * ptsDamage ) {
unsigned char const * pucFont = NULL;
int * piIndx = NULL;
int * piPos = NULL;
char ** argv = (char **)ppcData;
PhRect_t rect;
PhPoint_t pnt;
PhPoint_t tsPos = {0, 0};
PhRect_t tsExtent;
short n = 0;
char * pc = NULL;
PgColor_t old;

pucFont = szFont;
pc = argv[1];
piIndx = (int *)calloc(50, sizeof(int));
piPos = (int *)calloc(50, sizeof(int));

if (strlen(pc) < 4) {
    printf("Выберите строку подлиннее, она должна быть не менее 4 символов\n");
    exit(EXIT_SUCCESS);
}

for (n = 0; n < strlen(pc); n++) piIndx[n] = n + 1;

/* Находим наш холст */
PtBasicWidgetCanvas(pobjRaw, &rect);

old = PgSetStrokeColor(Pg_BLACK);

PfExtentText(&tsExtent, &tsPos, pucFont, pc, strlen(pc));

PgSetFont(pucFont);
PgSetTextColor(Pg_BLACK);

for (n = 0; n < strlen(pc); n++) piIndx[n] = n + 1;

/* Рисуем строку по символу за один раз */
PfExtentTextCharPositions(&tsExtent, &tsPos, pc, pucFont, piIndx, piPos,

```

```

        strlen(pc), 0L, 0, 0, NULL);
pnt.x = 10 + rect.ul.x;
pnt.y = 200 + rect.ul.y;

PgDrawIRect (tsExtent.ul.x + pnt.x,
             tsExtent.ul.y + pnt.y,
             (tsExtent.lr.x - min(tsExtent.ul.x, 0) + 1) + pnt.x,
             tsExtent.lr.y + pnt.y,
             Pg_DRAW_STROKE);

for (n = 0; n < strlen(pc); n++) {
    PgDrawText(pc + n, 1, &pnt, 0);
    pnt.x = 10 + rect.ul.x + piPos[n];
    printf("Один[%d]: %d\n", n, piPos[n]);
}
/* Конец рисования одного символа за раз */

/* Рисование строки, затем перекрытие отдельных символов
   сверху - справа налево */
printf("Проверка перекрытия\n");

PfExtentText(&tsExtent, &tsPos, pucFont, pc, strlen(pc));
pnt.x = 10 + rect.ul.x;
pnt.y = 400 + rect.ul.y;

PgDrawIRect (tsExtent.ul.x + pnt.x,
             tsExtent.ul.y + pnt.y,
             (tsExtent.lr.x - min(tsExtent.ul.x, 0) + 1) + pnt.x,
             tsExtent.lr.y + pnt.y,
             Pg_DRAW_STROKE);

PgSetFont (pucFont);
PgSetTextColor (Pg_BLACK);
PgDrawText (pc, strlen(pc), &pnt, 0);

for (n = strlen(pc) - 1; n >= 0; n--) {
    switch(n) {
        case 0:
            pnt.x = 10 + rect.ul.x;
            PgDrawText(pc + 0, strlen(pc), &pnt, 0);
            break;
        default:
            piIndx[0] = n;
            PfExtentTextCharPositions(&tsExtent, &tsPos, pc, pucFont, piIndx, piPos, 1, 0L, 0, 0,
            NULL);
            printf("Позиция: %d\n", piPos[0]);
            pnt.x = 10 + rect.ul.x + piPos[0];
            PgDrawText(pc + n, strlen(pc) - n, &pnt, 0);
            PgFlush();
            sleep(1);
            break;
    }
}

/* Завершаем рисования строки, затем перекрываем отдельные символы
   End draw string, then overlay individual characters
   сверху - справа налево */

PgSetStrokeColor (old);
free(piPos);
free(piIndx);

return( Pt_CONTINUE );
} // функция fnDrawCanvas()

```

Глава 20. Печать

Эта глава включает:

- Контекст печати
- Запуск процесса печати
- Печать требуемых виджетов
- Приостановка и возобновление процесса печати
- Прекращение процесса печати
- Освобождение контекста печати
- Пример

В Photon'e печать и прорисовка на экране – это одно и то же: разница зависит от контекста рисования – структуры данных, определяющих куда проходит поток рисования (т.е. события рисования):

- по умолчанию, на графический драйвер для рисования на экране
или
- на контекст памяти (или MC – memory context) для размещения в памяти для дальнейшего использования
или
- на контекст печати (или PC – printing context) для печати. См. раздел "Контекст печати" ниже.

Чтобы напечатать в Photon'e, необходимо:

1. Вызвав функцию PpCreatePC(), создать контекст печати
2. Выполнить установку контекста печати – либо автоматически через виджет PtPrintSel, либо программно с помощью функции PpSetPC().
3. Инициализировать процесс печати, вызвав функцию PpStartJob().
4. В любой момент после того, как функция PpStartJob() была вызвана, сделать контекст печати "активным", вызвав функцию PpContinueJob(). Когда контекст печати активен, всё, что рисуется через вызовы PpPrintWidget() или Pg*, включая виджеты, направляется в файл, открытый контекстом печати при вызове функции PpStartJob().
5. Вставить, если необходимо, принудительный обрыв страницы, вызвав функцию PpPrintNewPage().
6. Контекст печати можно сделать неактивным без прерывания текущего процесса печати, вызвав функцию PpSuspendJob(), или вызвав функцию PpContinueJob() с другим контекстом печати. Чтобы возобновить процесс печати с того места, где он был остановлен, необходимо вызвать функцию PpContinueJob().
7. Завершить процесс печати путём вызова функции PpEndJob().
8. Когда Вашему приложению больше не понадобится что-либо печатать в дальнейшем, вызвать функцию PpReleasePC(), чтобы освободить контекст печати.

Контекст печати

Контекст печати представляет из себя структуру типа PpContext_t, члены которой управляют тем, как выполняется печать. Информация о том, что находится в контексте печати, см. в "Справочнике библиотечных функций Photon'a".

- ☞ Никогда не обращайтесь напрямую к членам структуры PpPrintContext_t; используйте для извлечения членов функцию PpGetPC() и функцию PpSetPC() для их изменения.

Создание контекста печати

Первым шагом при выполнении печати в Photon'e является создание контекста печати с помощью функции PpCreatePC():

```
PpPrintContext_t pc;  
pc=PpCreatePC();
```

Модифицирование контекста печати

Сразу после того, как контекст создан, Вы должны выполнить его правильную установку в соответствии с Вашим принтером, и установить опции (ориентация, размер бумаги, прочая), которые Вы хотите использовать. Вы можете сделать это с помощью следующих функций:

- PpLoadDefaultPrinter()
- PpLoadPrinter()
- PpSetPC()
- PpPrintPropSelect()
- PpPrintSelect()
- PpPrintSelection()

Эти функции описаны в "Справочнике библиотечных функций Photon'a". Вы можете также использовать виджет PtPrintSel (см. "Справочник виджетов Photon'a").

Вы можете получить список доступных принтеров путём вызова функции PpLoadPrinterList(). Когда Вы завершите работу со списком, вызовите функцию PpFreePrinterList().

Запуск процесса печати

Если вы используете приложение, которому надо что-то знать о контексте печати, Вы можете использовать функцию PpGetPC(), с помощью которой можно получить соответствующую информацию. Например, Вам может понадобиться узнать о выбранной ориентации (для того, чтобы правильно сориентировать Ваши виджеты). Если Вам понадобится узнать размеры полей, Вы можете вызвать функцию PpGetCanvas().

Перед началом печати Вы должны установить размер или разрешение источника. Например:

- Если Вы хотите, чтобы виджет занял страницу, установите размер источника эквивалентным размерам виджета. Чтобы это сделать, можно вызвать функцию PpSetCanvas().
- По умолчанию разрешение источника равно 100 пикселям/дюйм, так что шрифты печатаются в приближённом размере. Вы можете получить размер внутреннего холста, вызвав функцию PpGetCanvas(), которая даёт размеры с учётом размеров полей и непечатаемых зон.

При установке размера источника учитывайте непечатаемые зоны принтера. Все принтеры имеют поля по краям страницы, на которых они не в состоянии печатать, даже если поля страниц установлены в 0. Поэтому размер, установленный выше, в действительности несколько больше, чем размер страницы, и шрифт будет масштабирован в сторону уменьшения, так чтобы поместиться в доступной для печати части листа.

В следующем примере размер страницы и непечатаемые области получены из расчёта предоставления надлежащего размера источника и высоты текста. Попробуйте выполнить это и оцените результирующий вывод, чтобы удостовериться, что шрифт имеет высоту в 1 дюйм от асцендера (верхнего элемента литеры) к десцендеру (подстрочному элементу литеры):

```

#include <stdio.h>
#include <stdlib.h>
#include <Pt.h>

PtWidget_t      *label, *window;
PpPrintContext_t *pc;

int quit_cb (PtWidget_t *widget, void *data, PtCallbackInfo_t *cbinfo ) {
    exit (EXIT_SUCCESS);
    return (Pt_CONTINUE);
} // функции quit_cb()

int print_cb (PtWidget_t *widget, void *data, PtCallbackInfo_t *cbinfo ) {
int action;
PhDim_t size;
PhRect_t const *rect;
PhDim_t const *dim;

action = PtPrintSelection(window, NULL, "Demo Print Selector", pc, Pt_PRINTSEL_DFLT_LOOK);
if (action != Pt_PRINTSEL_CANCEL) {
    /* Получение непечатаемой зоны и размера страницы, всё - в 1/1000 дюйма */

    PpGetPC(pc, Pp_PC_NONPRINT_MARGINS, &rect);
    PpGetPC(pc, Pp_PC_PAPER_SIZE, &dim);
    size.w = ((dim->w - (rect->ul.x + rect->lr.x)) * 72) / 1000;
    size.h = ((dim->h - (rect->ul.y + rect->lr.y)) * 72) / 1000;

    /* Установка размера источника */
    PpSetPC( pc, Pp_PC_SOURCE_SIZE, &size, 0);

    /* Начало печати надписи */
    PpStartJob(pc);
    PpContinueJob(pc);

    /* Повреждение виджета */
    PtDamageWidget(label);
    PtFlush();

    /* Закрытие печати */
    PpSuspendJob(pc);
    PpEndJob(pc);
}
return (Pt_CONTINUE);
} // функции print_cb()

int main(int argc, char *argv[]) {
PtArg_t args[10];
PtWidget_t *print, *quit;
PhDim_t win_dim = { 400, 200 };
PhPoint_t lbl_pos = {0, 0};
PhArea_t print_area = { {130, 170}, {60, 20} };
PhArea_t quit_area = { {210, 170}, {60, 20} };
PtCallback_t callbacks[2] = { {print_cb, NULL}, {quit_cb, NULL} };

if (PtInit(NULL) == -1) PtExit(EXIT_FAILURE);

/* Создание главного окна */
PtSetArg (&args[0], Pt_ARG_DIM, &win_dim, 0);
PtSetArg (&args[1], Pt_ARG_WINDOW_TITLE, "Пример печати", 0);

if ((window = PtCreateWidget(PtWindow, Pt_NO_PARENT, 1, args)) == NULL)
PtExit (EXIT_FAILURE);

/* Создание контекста печати */
pc = PpCreatePC();

/* Создание надписи для печати */
PtSetArg (&args[0], Pt_ARG_POS, &lbl_pos, 0);
PtSetArg (&args[1], Pt_ARG_TEXT_STRING, "Моя высота 1 дюйм", 0);
PtSetArg (&args[2], Pt_ARG_TEXT_FONT, "swiss72", 0);
PtSetArg (&args[3], Pt_ARG_MARGIN_HEIGHT, 0, 0);
PtSetArg (&args[4], Pt_ARG_MARGIN_WIDTH, 0, 0);
PtSetArg (&args[5], Pt_ARG_BEVEL_WIDTH, 0, 0);
label = PtCreateWidget (PtLabel, window, 6, args);

/* Создание кнопки печати */
PtSetArg(&args[0], Pt_ARG_AREA, &print_area, 0);
PtSetArg(&args[1], Pt_ARG_TEXT_STRING, "Печать", 0);

```

```

PtSetArg(&args[2], Pt_CB_ACTIVATE, &callbacks[0], 0);
print = PtCreateWidget (PtButton, window, 3, args);

/* Создание кнопки завершения */
PtSetArg(&args[0], Pt_ARG_AREA, &quit_area, 0);
PtSetArg(&args[1], Pt_ARG_TEXT_STRING, "Конец", 0);
PtSetArg(&args[2], Pt_CB_ACTIVATE, &callbacks[1], 0);
quit = PtCreateWidget (PtButton, window, 3, args);

PtRealizeWidget(window);
PtMainLoop();
return (EXIT_SUCCESS);
} // main()

```

Вы должны также установить смещение источника – верхний левый угол того, что печатается. Например, если у Вас есть кнопка, нарисованная на (20,20) от верхнего левого угла панели и Вы хотите, чтобы на странице она рисовалась на (0,0), установите в (20,20) смещение источника. Любые другие виджеты рисуются на своих позициях относительно этого начального значения виджетов. Виджет, расположенный на (40,40), будет прорисован на странице в позиции (20,20). Код для этого следующий:

```

PhPoint_t offset = {20, 20};
...
PpSetPC( pc, Pp_PC_SOURCE_OFFSET, &offset, 0 );

```

Сразу же после установки размеров источника и смещения, Вы можете запустить печать:

```

PpStartJob(pc);
PpContinueJob(pc);

```

Функция PpStartJob(pc) устанавливает контекст печати для процесса печати и функция PpContinueJob(pc) делает контекст печати активным, в результате чего все команды рисования Photon'a перенаправляются на адресата, заданного в контексте печати.

Печать требуемых виджетов

После того как Вы сделали контекст печати активным, Вы можете запустить печать виджетов и всё такое прочее. Это может быть сделано путём вызова любой комбинации следующих функций:

- Функции Pg*
- Функция PpPrintWidget() – Вы можете напечатать даже виджет, который не был нереализован (unrealise).

☞ Если вы можете напечатать всё содержание скроллируемого виджета, Вам понадобится выполнить определённые подготовительные операции. См. раздел "Печать скроллирующихся виджетов" ниже.

Печать новой страницы

Вы можете в любой точке выполнить принудительный конец страницы, вызвав функцию PpPrintNewPage():

```

PpPrintNewPage(pc);

```

Заметьте, что как только Вы вызвали функцию PpStartJob(), какие-либо изменения контекста печати будут иметь эффект только после следующего вызова функции PpPrintNewPage().

Photon предполагает, что номера страниц увеличиваются по 1. Если это не так, вручную установите член Pp_PC_PAGE_NUM контекста печати для установки нужного номера страницы.

Не делайте номер страницы уменьшающимся, так как в этом случае драйверы печати могут не работать должным образом.

Печать скроллирующихся виджетов

Если Вы хотите напечатать всё содержание скроллирующегося виджета, Вам надо выполнить некую особую обработку.

PtList

Единственным способом заставить PtList напечатать (или нарисовать) все пункты – это изменить его размеры так, чтобы высота была равна общей высоте всех пунктов. Простейшим решением этого является, вероятно, использование политики изменения размеров:

☞ Это будет работать только в том случае, если общая высота меньше чем 65К пикселей.

1. Открыть и запустить контекст печати.
2. Получить для виджета PtList текущие значения флагов изменения размеров (Pt_ARG_RESIZE_FLAGS).
3. Установить флаги изменения размеров в значение Pt_RESIZE_XY_ALWAYS, в результате изменяя размер списка так, чтобы он вместил весь свой текст.
4. Вызвать функцию PpPrintWidget() для виджета или родительского виджета.
5. Переустановить флаги изменения размеров для виджета PtList.
6. Остановить и закрыть контекст печати.

PtMultiText

Из-за программного дефекта в флагах изменения размеров в многострочном виджете, метод, используемый для виджета PtList, работает неверно с виджетом PtMultiText.

Чтобы напечатать весь текст виджета PtMultiText:

1. Откройте и запустите контекст печати.
2. Получите текущие значения ресурсов Pt_ARG_MULTITEXT_NUM_LINES и Pt_ARG_MULTITEXT_NUM_LINES_VISIBLE виджета.
3. Сохраните значение ресурса Pt_ARG_MULTITEXT_NUM_LINES_VISIBLE в локальной переменной (помните, что PtGetResources() даёт Вам указатель во внутренней памяти виджета – не рассчитывайте, что в нём можно хранить число видимых строк).
4. Установите Pt_ARG_MULTITEXT_ROWS в значение Pt_ARG_MULTITEXT_NUM_LINES.
5. Вызовите PpPrintWidget() для виджета или родительского виджета.
6. Переустановите значение Pt_ARG_MULTITEXT_ROWS, чтобы в нём хранилось число видимых строк.
7. Остановите и закройте контекст печати.

Вот ответная реакция, которая распечатывает виджет PtMultiText:

```
int prt_multi( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo ) {
int action;
PhDim_t size = { 850, 1100 };
long *num_lines;
short *num_lines_visible;
long vis_lines;
PtArg_t args[2];

/* предотвращает предупреждения (варнинги) об отсутствии ссылок */
widget = widget, apinfo = apinfo, cbinfo = cbinfo;

/* Вы можете выполнить эти вызовы PpSetPC() просто после создания контекста печати
Его наличие позволяет Вам повторно использовать контекст печати */

/* Установка разрешения источника пропорционально размеру страницы */
```

```

PpSetPC( pc, Pp_PC_SOURCE_SIZE, &size, 0 );

action = PtPrintSelection(ABW_base, NULL, "Демонстрационный селектор печати", pc,
                          Pt_PRINTSEL_DFLT_LOOK);

if (action != Pt_PRINTSEL_CANCEL) {
    /* Запуск печати панели окна. Заметьте, что мы используем
       один и тот же контекст печати для всех печатных работ */
    PpStartJob(pc);
    PpContinueJob(pc);

    /* Получение числа строк и количества видимых строк */

    PtSetArg (&args[0], Pt_ARG_MULTITEXT_NUM_LINES, &num_lines, 0);
    PtSetArg (&args[1], Pt_ARG_MULTITEXT_NUM_LINES_VISIBLE, &num_lines_visible, 0);
    PtGetResources (ABW_multi, 2, args);

    /* Сохранение числа видимых строк в локальной переменной;
       помните, что num_lines_visible указывает на внутреннюю память виджета */

    vis_lines = *num_lines_visible;

    /* Установка числа строк равным числу строчек текста */

    PtSetResource( ABW_multi, Pt_ARG_MULTITEXT_ROWS, *num_lines, 0);

    /* Печать виджета */
    PpPrintWidget(pc, ABW_multi, NULL, NULL, 0);

    /* Закрытие контекста печати */
    PpSuspendJob(pc);
    PpEndJob(pc);

    /* Переустановка числа строк в сохранённое значение количества видимых строчек */

    PtSetResource (ABW_multi, Pt_ARG_MULTITEXT_ROWS, vis_lines, 0);
}

return( Pt_CONTINUE );
}

```

PtScrollArea

Для PtScrollArea Вам необходимо печатать его виртуальный холст, в котором размещены все виджеты, созданные внутри него или перемещаемые по зоне прокрутки:

1. Получите указатель на виртуальный холст, вызвав:

```
PtValidParent(ABW_Scroll_area, PtWidget);
```

2. Получите зону Pt_ARG_AREA) виртуального холста, и используйте его член *member* как размер источника в контексте печати;

3. Установите смещение контекста печати источника в:

```
PtWidgetOffset(PtValidParent(ABW_Scroll_area, PtWidget));
```

4. Напечатайте виртуальный холст области прокрутки, вызвав

```
PpPrintWidget(pc, PtValidParent(ABW_Scroll_area, PtWidget), NULL, NULL, 0);
```

Приостановка и возобновление работы печати

Чтобы приостановить работу печати и направить все события рисования обратно на графический драйвер в любой момент после вызова PpStartJob(), вызовите

```
PpSuspendJob(pc);
```

Чтобы возобновить работу печати, вновь активизировав контекст печати, так чтобы направить события прорисовки в направлении адресата, заданного в контексте печати, вызовите

```
PpContinueJob (pc);
```

Завершение работы печати

Когда Вы завершили печать Ваших виджетов, необходимо деактивировать и закрыть контекст печати. Это выполняется вызовами:

```
PpSuspendJob (pc);  
PpEndJob (pc);
```

Все события прорисовки будут направлены на графический драйвер.

- ☞ Вы можете повторно использовать контекст печати для новых работ печати, что позволяет не создавать и не инициализировать его вновь.

Освобождение контекста печати

Когда печать завершена и Вам больше не нужен контекст печати, Вы можете его освободить, что освободит все используемые им ресурсы.

Если Вы хотите запомнить какую-либо информацию из контекста печати для дальнейшего использования, сохраните её, вызвав функцию PpGetPC() перед тем, как освободить контекст печати. Например:

```
short const *orientation;  
...  
PpGetPC ( pc, Pp_PC_ORIENTATION, &orientation );  
Чтобы освободить контекст печати, вызовите:  
PpReleasePC ( pc );
```

Пример

В этом примере создаётся приложение с главным окном и панелью окна с несколькими виджетами на ней. Когда Вы нажимаете кнопку "Печать", появляется диалог селектора печати. Когда Вы выбираете в этом диалоге кнопку "Print" или "Preview", панель "рисуеться" на принтере.

```
#include <stdio.h>  
#include <stdlib.h>  
#include <Pt.h>  
  
PtWidget_t *pane, *window;  
PpPrintContext_t *pc;  
  
int quit_cb ( PtWidget_t *widget, void *data, PtCallbackInfo_t *cbinfo) {  
PpReleasePC (pc);  
exit (EXIT_SUCCESS);  
return (Pt_CONTINUE);  
} // функции quit_cb()
```

```

int print_cb ( PtWidget_t *widget, void *data, PtCallbackInfo_t *cbinfo) {
int action;

/* Вы можете выполнить эти вызовы PpSetPC() просто после создания контекста печати
Его наличие позволяет Вам повторно использовать контекст печати */
PhDim_t size = { 850, 1100 };
PhDim_t size2 = { 200, 150 };

/* Установка разрешения источника пропорционально размеру страницы */
PpSetPC(pc, Pp_PC_SOURCE_SIZE, &size, 0);

/* Раскомментируйте это, чтобы установить размеры источника равными размерам виджета.
При печати виджет будет масштабирован */
/* PpSetPC(pc, Pp_PC_SOURCE_SIZE, &size2, 0); */

action = PtPrintSelection(window, NULL, " Демонстрационный селектор печати ", pc,
Pt_PRINTSEL_DFLT_LOOK);
if (action != Pt_PRINTSEL_CANCEL) {
/* Запуск печати панели окна. Заметьте, что мы используем
один и тот же контекст печати для всех печатных работ */
PpStartJob(pc);
PpContinueJob(pc);

/* Печать виджета */
PpPrintWidget(pc, pane, NULL, NULL, 0);

/* Закрытие контекста печати */
PpSuspendJob(pc);
PpEndJob(pc);
}

return (Pt_CONTINUE);
} // функции print_cb()

int main(int argc, char *argv[]) {
PtArg_t args[4];
PtWidget_t *print, *quit;
PhDim_t win_dim = { 200, 200 };
PhArea_t pane_area = { {0, 0}, {200, 150} };
PhArea_t print_area = { {30, 170}, {60, 20} };
PhArea_t quit_area = { {110, 170}, {60, 20} };
PhArea_t cir_area = { {35, 20}, {130, 110} };
PhArea_t cir2_area = { {67, 40}, {20, 20} };
PhArea_t cir3_area = { {110, 40}, {20, 20} };
PhArea_t cir4_area = { {85, 80}, {30, 30} };
PtCallback_t callbacks[2] = { {print_cb, NULL}, {quit_cb, NULL} };

if (PtInit(NULL) == -1) PtExit(EXIT_FAILURE);

/* Создание основного окна */
PtSetArg (&args[0], Pt_ARG_DIM, &win_dim, 0);
PtSetArg (&args[1], Pt_ARG_WINDOW_TITLE, "Пример печати", 0);
if ((window = PtCreateWidget(PtWindow, Pt_NO_PARENT, 2, args)) == NULL)
PtExit(EXIT_FAILURE);

/* Создание контекста печати */
pc = PpCreatePC();

/* Создание панели, которая будет напечатана */
PtSetArg (&args[0], Pt_ARG_AREA, &pane_area, 0);
pane = PtCreateWidget (PtPane, window, 1, args);

/* Помещаем на панель какую-то ерунду для её распечатывания */
PtSetArg (&args[0], Pt_ARG_AREA, &cir_area, 0);
PtCreateWidget (PtEllipse, pane, 1, args);

PtSetArg (&args[0], Pt_ARG_AREA, &cir2_area, 0);
PtSetArg (&args[1], Pt_ARG_FILL_COLOR, Pg_BLACK, 0);
PtCreateWidget (PtEllipse, pane, 2, args);

PtSetArg (&args[0], Pt_ARG_AREA, &cir3_area, 0);
PtSetArg (&args[1], Pt_ARG_FILL_COLOR, Pg_BLACK, 0);
PtCreateWidget (PtEllipse, pane, 2, args);

PtSetArg (&args[0], Pt_ARG_AREA, &cir4_area, 0);
PtCreateWidget (PtEllipse, pane, 1, args);

```

```
/* Создание кнопки печати */
PtSetArg(&args[0], Pt_ARG_AREA, &print_area, 0);
PtSetArg(&args[1], Pt_ARG_TEXT_STRING, "Печать", 0);
PtSetArg(&args[2], Pt_CB_ACTIVATE, &callbacks[0], 0);
print = PtCreateWidget (PtButton, window, 3, args);

/* Создание кнопки завершения */
PtSetArg(&args[0], Pt_ARG_AREA, &quit_area, 0);
PtSetArg(&args[1], Pt_ARG_TEXT_STRING, "Конец", 0);
PtSetArg(&args[2], Pt_CB_ACTIVATE, &callbacks[1], 0);
quit = PtCreateWidget (PtButton, window, 3, args);

PtRealizeWidget(window);
PtMainLoop();
return (EXIT_SUCCESS);
}
```

Глава 21. "Тащить и бросать"

Технология "тащить и бросать" (drag&drop) позволяет Вам перетаскивать произвольные данные внутри приложения или между приложениями.

☞ Если Вам надо просто перетаскивать графические объекты, см. раздел "Перетаскивание" в главе "События".

В этой главе обсуждается:

- Механизм транспортировки
- Использование "тащить и бросать"
- Регистрация новых транспортных типов

Механизм транспортировки

Механизм транспортировки в Photon'e позволяет Вам переносить любые данные из одного приложения в другое, даже если приложения принадлежат к различным платформам с отличающимся форматом передачи данных (расположением старшего/младшего байтов). Этот механизм используется как основа технологии "тащить и бросать", но может использоваться и для других целей, таких как конфигурационные файлы.

Имеется два способа транспортировки данных:

Inline Поточный. Данные пакуются в поток (stream) и отсылаются адресату.
By request По запросу. Пакуются в поток и отсылаются описания данных. Адресат принимает решение, какой тип (типы) данных ему нужен и отправляет запрос обратно на источник, который затем упаковывает только затребованные данные.

Для того, чтобы транспортировать данные, механизм транспортировки должен паковать данные в источнике – приложении или виджете – и распаковывать их в адресате. Он должен быть способен опознавать тип данных, чтобы определять, какой вид паковки и распаковки должен быть выполнен. Это выполняется через *транспортный реестр*.

Существует несколько регистрируемых системой типов, которые появляются после инициализации Photon'овской библиотеки через вызовы PtInit() или PtAppInit() – это для приложений PhAB выполняется автоматически. Регистрируемыми системой типами являются:

- string
- raw
- PhDim
- PhArea
- PhPoint
- PhImage

Вы можете добавить в реестр другие типы данных, как это описано в разделе "Регистрация новых транспортных типов" ниже в этой главе. Механизм транспортировки работает посредством построения списка данных, предназначенных для транспортировки, пакования данных в поток, в котором каждому блоку предшествует заголовок, описывающий данные.

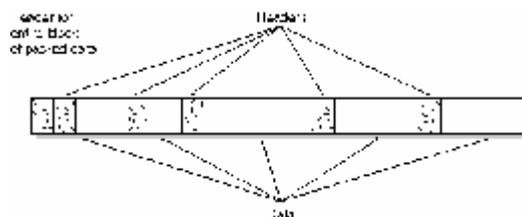


Рис. 21-1. Упакованные данные и заголовки

Когда данные прибывают к адресату, заголовки извлекаются для получения распаковочных инструкций к данным. Механизм транспортировки автоматически распаковывает данные; приложение получает данные в своём оригинальном виде.

Использование "тащи и бросай"

Этот раздел включает:

- Запуск "тащи и бросай"
- Получение событий по "тащи и бросай"
- Отмена "тащи и бросай"

Вы можете использовать "тащи и бросай", чтобы переместить данные из одного виджета в другой, используя транспортный механизм Photon'a. Вы можете транспортировать одновременно данные нескольких типов, предоставляя адресату выбор, какие данные получать. Весь обмен между источниками и адресатом является неблокирующим.

Основными шагами (описанными более подробно в нижеследующих разделах) являются:

1. Пользователь нажимает кнопку указателя на виджете, который в операции "тащи и бросай" является источником.
2. В ответной реакции `Pt_CB_OUTBOUND` виджет-источник пакует данные, предназначенные для перетаскивания, и запускает операцию "тащи и бросай".
3. Пользователь перетаскивает данные и рашает бросить их на виджет.
4. В ответной реакции `Pt_CB_DND` виджет-адресат решает, какую часть перетасканных данных (если таковая имеется) он примет. Все доступные данные распаковываются автоматически. Данные размещаются в выделенной памяти, адресат должен освободить память, когда данные ему больше не нужны.

Виджет-источник может также, если захочет, отменить операцию.

Запуск операции "тащи и бросай"

Чтобы запустить операцию "тащи и бросай", виджет-источник должен запаковать данные, которые будут перетаскиваться, и затем инициализировать операцию "тащи и бросай". Обычно это делается в одной из ответных реакций `Pt_CB_OUTBOUND` виджета.

☞ Ответные реакции `Pt_CB_OUTBOUND` вызываются только в случае, когда у виджета в его флагах `Pt_ARG_FLAGS` установлен флаг `Pt_SELECTABLE`.

Выполните следующие шаги:

1. Если данные не принадлежат к одному из определённых системой типов транспортных данных, создайте для них член транспортного реестра (transport registry entry). Более подробно см. в разделе "Регистрация новых транспортных типов" ниже.
2. Создайте структуру транспортного управления (типа `PtTransportCtrl_t`) для использования операцией "тащи и бросай", вызвав функцию `PtCreateTransportCtrl()`. Структура транспортного управления освобождается автоматически после исполнения операции "тащи и бросай".
3. Данные для перетаскивания могут быть запакованы поточно (т.е. включены непосредственно в структуру, передаваемую адресату) или же они могут быть данными, получаемыми по запросу (т.е. данные не запаковываются до тех пор, пока адресат их не запросит).
 - Для каждого фрагмента данных, пакуемых поточно, вызовите функцию `PtTransportType()`.
 - Для каждого фрагмента данных, передаваемых по запросу, вызовите функцию `PtTransportRequestable()`.

Структура `PtTransportCtrl_t` имеет список запрашиваемых данных, который автоматически отсылается, если адресат их запрашивает. Виджет-источник может добавить данные в эту очередь, вызвав функцию `PtAddResponseType()`.

Если виджет-источник в этот момент не желает паковать данные, доступные по запросу, это должно обеспечиваться ответной реакцией при вызове PtTransportRequestable().

4. Когда все данные запакованы, вызовите функцию PtInitDnd(), чтобы инициализировать операцию "тащи и бросай".

Пример

Вот пример ответной реакции, инициализирующей операцию "тащи и бросай" для виджета типа PtLabel. Вы можете использовать эту ответную реакцию для ответной реакции Pt_CB_OUTBOUND виджетов типа PtLabel (виджетов-надписей).

☞ Убедитесь, что во флагах Pt_ARG_FLAGS виджета установлен флаг Pt_SELECTABLE.

Эта ответная реакция устанавливает операцию "тащи и бросай", включая следующие фрагменты данных:

- Текст надписи, если он имеется, пакуется как потоковые данные.
- Образ надписи, если он имеется, пакуется как данные, доступные по запросу. Образ не транспортируется до тех пор, пока адресат не запросит его, но ответная реакция заблаговременно готовит эти данные, так что нам не требуется ответная реакция запроса.
- Альтернативный текст, используемый при отсутствии образа. Эта строка также представляет собой данные, доступные по запросу, но мы предусматриваем ответную реакцию для упаковки этих данных, если адресат их запросит.

Ответная реакция устанавливает некий групповой номер для образа и альтернативного текста, чтобы указать, что они представляют собой различные формы одних и тех же данных.

```

/* Стандартные хедеры */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

/* Инструментальные хедеры */
#include <Ph.h>
#include <Pt.h>
#include <Ap.h>

/* Локальные хедеры */
#include "abimport.h"
#include "proto.h"

#define TEXT_GROUP 0
#define IMAGE_GROUP 1

PtTransportReqDataCB_t request_callback;

int start_dnd( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo ) {

char *widget_text = NULL;
char *label_type;
PhImage_t * image = NULL;
PtRequestables_t *req;
PtTransportCtrl_t *tctrl = PtCreateTransportCtrl();
int ret;

/* предотвращает предупреждения (варнинги) об отсутствии ссылок */
widget = widget, apinfo = apinfo;
cbinfo = cbinfo;

/* Получает тип надписи, так что мы можем определить, какие данные паковать */
PtGetResource( widget, Pt_ARG_LABEL_TYPE, &label_type, 0);

if ((*label_type == Pt_Z_STRING) || (*label_type == Pt_TEXT_IMAGE)) {
/* Получает текст виджета и пакует его в поток */
PtGetResource( widget, Pt_ARG_TEXT_STRING, &widget_text, 0);
PtTransportType( tctrl, "text", "plain", TEXT_GROUP, Ph_TRANSPORT_INLINE, "string",
widget_text, 0, 0);
}

/* Если это образ, добавляем его как данные, доступные по запросу.
Готовим запрашиваемые данные (позволяя автоматический запрос)
*/

```

```

if ((*label_type == Pt_IMAGE) || (*label_type == Pt_TEXT_IMAGE)) {
    PtGetResource( widget, Pt_ARG_LABEL_IMAGE, &image, 0);
    if (image) {
        req = PtTransportRequestable ( tctrl, "image", "an image", IMAGE_GROUP,
                                     Ph_TRANSPORT_INLINE, "PhImage", NULL, NULL );
        PtAddResponseType( tctrl, req, "image", "an image", Ph_TRANSPORT_INLINE, "PhImage",
                           image, 0, 0);
    }
}

/* Добавляем доступную по запросу строку, которая будет
   предоставлена по ответной реакции
*/

PtTransportRequestable( tctrl, "text", "image description", IMAGE_GROUP,
                       Ph_TRANSPORT_INLINE, "string",
                       (PtTransportReqDataCB_t *) &request_callback, "This was requested");

/* Инициализируем операцию "тащи и бросай" */

ret = PtInitDnd( tctrl, widget, cbinfo->event, NULL, 0);
return( Pt_CONTINUE );
} // функции start_dnd()

int unsigned request_callback( int unsigned type, PtReqResponseHdr_t *req_hdr,
                              PtRequestables_t *req) {
    if (type == Pt_DND_REQUEST_DATA) {
        /* Ответить на запрос строкой из req->rq_callback_data,
           последнего аргумента в PtTransportRequestable()
        */

        PtAddResponseType( req->ctrl, req, "text", "request", Ph_TRANSPORT_INLINE, "string",
                           req->rq_callback_data, 0, 0);

        return Pt_CONTINUE;
    }

    /* Отвергнуть запрос */
    return Pt_END;
} // функции request_callback()

```

Получение событий "тащи и бросай"

Чтобы виджет был в состоянии получать события "тащи и бросай", прикрепите к нему ответную реакцию Pt_CB_DND (см. описание PtWidget в "Справочнике виджетов Photon'a").

☞ Чтобы у виджета вызывались его ответные реакции Pt_CB_DND, в его флагах Pt_ARG_FLAGS не требуется устанавливать флаг Pt_SELECTABLE.

Всякий раз, когда каким-либо образом виджет вовлекается в событие "тащи и бросай", вызывается его ответная реакция Pt_CB_DND. В ответной реакции *cbinfo->reason_subtype* указывает тип произошедшего действия "тащи и бросай".

В нижеследующем разделе описаны события операции "Тащи и бросай", интересующие виджет-источник и виджет-адресат. Конечно, если виджет может быть и источником, и адресатом в (отдельных) операциях "тащи и бросай", его ответные реакции Pt_CB_DND должны иметь оба набора событий. Более подробно информация о событиях дана в описании типа PhEvent_t "Справочника библиотечных функций Photon'a".

Виджет-источник

Виджет-источник операции "тащи и бросай" может получать события, описывающие состояние операции. Если Вам не нужны эти события, установите флаг Pt_DND_SILENT в аргументе *flags* функции PtInitDnd().

☞ Не устанавливайте флаг Pt_DND_SILENT, если Вы включаете доступные по запросу данные в структуру управления.

Подтипами события операции "тащи и бросай", представляющими интерес для источника операции, являются:

Ph_EV_DND_INIT	Операция успешно запущена
Ph_EV_DND_CANCEL	Операция была отменена (например, если сбрасывание произошло не над зоной, где возможен сброс, или адресат прервал операцию до получения сброса, или до того, как завершил выборку запрашиваемых данных). Если операция отменена таким образом, библиотека автоматически очищает структуры данных.
Ph_EV_DND_COMPLETE	Событие операции "тащи и бросай" поставлено адресатом в очередь (адресат пока что его не рассматривает).
Ph_EV_DND_DELIVERED	Адресат удалил событие операции "тащи и бросай" из очереди.

Виджет-адресат

Подтипами события "тащи и бросай", представляющими интерес для адресата операции, являются:

Ph_EV_DND_ENTER	Кто-то перетащил некие данные в область виджета, но ещё не сбросил их. Этот подтип события (<i>reason_subtype</i>) является первоначальной причиной вызова ответной реакции операции "тащи и бросай". В этот момент приложение принимает решение, примет ли оно сброшенные данные. Оно должно построить некий массив структур типа <i>PtDndFetch_t</i> и передать его функции <i>PtDndSelect()</i> . Это массив описывает допустимые типы, описания и транспортные методы для данных, участвующих в операциях "тащи и бросай", доступные для виджета. Функция <i>PtDndSelect()</i> возвращает число выбранных элементов из массива. Если событие содержит данные или ссылки на данные, в допустимом формате, выбираются и эти фрагменты событий. Если никакие данные не допустимы, этот виджет никакими другими событиями текущей операции "тащи и бросай" не модифицируется.
Ph_EV_DND_MOTION	Указатель перемещается внутри зоны виджета. Этот тип событий генерируется только в случае, когда для фрагмента выбранных данных в члене <i>select_flags</i> структуры <i>PtDndFetch_t</i> установлен бит <i>Pt_DND_SELECT_MOTION</i> .
Ph_EV_DND_DROP	Пользователь сбросил данные. Для этого подтипа события ответная реакция получает из события выбранные данные. Это может включать в себя какую-то автоматическую, неблокируемую связь с источником данных – чтобы не допустить какую-либо связь с источником, задайте <i>Ph_TRANSPORT_INLINE</i> в качестве единственно допустимого транспортного протокола. Если сброс данных выполнен успешно, память, использовавшаяся транспортным механизмом, автоматически освобождается.
Ph_EV_DND_LEAVE	Указатель вышел за пределы зоны виджета, но пользователь не сбросил данные.

Вот пример, работающий с ответной реакцией, приведённой выше для виджета *PtLabel*. Эта ответная реакция допускает в качестве данных операции "тащи и бросай" следующие типы:

- текст
- образ
- альтернативная строка, если в данных отсутствует образ (в элементе *PtDndFetch_t* установлен *Pt_DND_SELECT_DUP_DATA*).

Виджет-источник пакует образ и альтернативную строку как данные, доступные по запросу, но адресат не предпринимает ничего, чтобы их затребовать; механизм транспортировки делает это автоматически.

```

/* Стандартные хеадеры */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

/* Инструментальные хеадеры */
#include <Ph.h>
#include <Pt.h>
#include <Ap.h>

/* Локальные хеадеры */
#include "abimport.h"
#include "proto.h"

static PtDndFetch_t stuff_i_want[] = {
    {"text", "plain", Ph_TRANSPORT_INLINE, },
    {"image", NULL, Ph_TRANSPORT_INLINE, },
    {"text", "image description", Ph_TRANSPORT_INLINE, Pt_DND_SELECT_DUP_DATA, },
};

enum {
    PLAIN_TEXT = 0,
    IMAGE,
    IMAGE_TEXT,
};

int dnd_callback( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo ) {
    PtDndCallbackInfo_t *dndcb = cbinfo->cbdata;
    int deep_free = 1, num_matches;

    /* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    widget = widget, apinfo = apinfo;
    cbinfo = cbinfo;

    switch (cbinfo->reason_subtype) {
        case Ph_EV_DND_ENTER:
            num_matches = PtDndSelect (widget, stuff_i_want, ARRAY_SIZE( stuff_i_want ),
                                     NULL, NULL, cbinfo );
            break;

        case Ph_EV_DND_DROP:
            switch (dndcb->fetch_index) {
                case PLAIN_TEXT:
                    PtSetResource (widget, Pt_ARG_TEXT_STRING, dndcb->data, strlen(dndcb->data));
                    break;

                case IMAGE:
                    PtSetResource (widget, Pt_ARG_LABEL_IMAGE, dndcb->data, 0);
                    free (dndcb->data);
                    deep_free = 0;
                    break;

                case IMAGE_TEXT:
                    printf ("Образ отсутствует; альтернативный текст: %s\n", (char *)dndcb->data);
                    break;
            }

            if (deep_free) {
                PhFreeTransportType (dndcb->data, dndcb->trans_hdr->packing_type);
            }
            break;
    }
    // switch по подтипу вызвавшего события

    return( Pt_CONTINUE );
}

```

Отмена операции "тащи и бросай"

Виджет-источник может отменить операцию "тащи и бросай", вызвав `PtCancelDnd()`. Виджет должен затем очистить структуры управления транспортом и запакованные данные, вызвав функцию `PtReleaseTransportCtrl()`. (Если сброс был выполнен успешно, управляющие структуры очищаются автоматически. Адресат решает, когда ему освободить сброшенные данные).

Регистрация новых транспортных типов

В этом разделе обсуждается:

- Простая структура данных
- Более сложная структура
- Транспортные функции

Чтобы транспортировать данные типов, отличных от тех, которые автоматически определяются Photon'ом, Вы должны определить тип и зарегистрировать его в *транспортном реестре* – наборе описаний типов, каждое из которых включает:

- имя типа в виде строки, например, `PhImage`
- метод упаковки, который будет использоваться (один из следующих: `Ph_PACK_RAW`, `Ph_PACK_STRING` или `Ph_PACK_STRUCT`)
- список членов внутри типа, которые ссылаются на данные в запросе вне базового размера типа (ссылка или члены типа указателя)
- список членов, которые чувствительны к способу передачи данных (вперёд старшим байтом или назад) (эти члены являются корректными при передаче, даже если распаковываются на машине, способ передачи данных у которой отличается от способа на той машине, где данные были упакованы)
- список членов, которые должны быть очищены, когда тип распакован (например, указатель на данные – такие как пароль – которые Вы не хотите транспортировать).

☞ Перед тем, как данные могут быть успешно транспортированы, тип данных должен быть определён в реестрах транспортировки и в приложении-источнике, и в приложении-адресате.

Простая структура данных

Давайте рассмотрим простую структуру данных:

```
typedef struct simpl {
    int num;
    int nums_10[10];
    char name[10];
    short vals_5[5];
} Simpl_t;
```

Эту структуру можно легко запаковать, используя тип *raw*, поскольку она не содержит каких-либо внешних ссылок (т.е. не имеет членов-указателей). Но это не защищает транспортируемые данные от различий между источником и адресатом, заключающихся в способе передачи данных (вперёд старшим или младшим байтом). [Так, достало меня это писать! Дальше по тексту этот "способ передачи данных вперёд старшим или младшим байтом" "первожу" как эндиан. Прим. пер.] Так что даже для такой простой структуры полезно описание типа, детализирующее его эндиан-чувствительность.

Описание типа начинается с массива элементов типа `init unsigned`, которые описывают эндиан-чувствительность для каждого члена:

```
static const int unsigned SimplEndians[] = {
    Tr_ENDIAN( Simpl_t, num ),
    Tr_ENDIAN_ARRAY( Simpl_t, nums_10 ),
    Tr_ENDIAN_ARRAY( Simpl_t, vals_5 ),
    0 /* Конец эндиан-списка */
};
```

Обратите внимание, что этот список должен завершаться нулевым элементом. Член *name* не является эндиан-чувствительным, поэтому он не включён в список.

Все типы или ссылки на типы устанавливают эндиан-параметры для своих членов на основе эндиан-массива, описанного в типе. Классификация эндиан-чувствительных членов следующая:

```
Tr_ENDIAN( typedef_name, member )
    int, long, short, (signed или unsigned).
```

Например, `unsigned int my_scalar`.

`Tr_ENDIAN_ARRAY(typedef_name, member)` - Массив из `short` или `int` элементов.

Например, `short short_nums[10]`.

`Tr_ENDIAN_REF(typedef_name, member, num)` - Ссылка на эндиан-скаляры. Например, `int *nums`.

Имея для нашего простого типа данных задание эндиан-списка, давайте создадим определение, чтобы обратиться в транспортный реестр:

```
static const PhTransportRegEntry_t SimplTransDef = {
    "simpl",
    Ph_PACK_STRUCT,
    sizeof( Simpl_t ),
    0,
    NULL,
    &SimplEndians,
    NULL
};
```

Структура `PhTransportRegEntry_t` включает следующие члены:

<code>char*type</code>	Имя регистрируемого типа
<code>int unsigned packing</code>	Метод установки, который будет использоваться (один из: <code>Ph_PACK_RAW</code> , <code>Ph_PACK_STRING</code> или <code>Ph_PACK_STRUCT</code>)
<code>int unsigned size</code>	Размер типа данных в байтах
<code>int unsigned num_fixups</code>	Количество элементов в массиве <code>fixups</code>
<code>PhTransportFixupRec_t const *fixups</code>	Список инструкций по работе со ссылками на данные, определённые вне типа. Подробнее будет обсуждено далее
<code>int unsigned const *endians</code>	Завершающийся нулём массив эндиан-информации, описанный выше
<code>int unsigned const *clear_refs</code>	Завершающийся нулём массив членов, которые должны быть очищены (т.е. установлены в <code>NULL</code>), когда данные этого типа распакованы

Чтобы зарегистрировать этот впервые определённый тип, вызовите функцию `PhRegisterTransportType()`:

```
PhRegisterTransportType(&Simpl, TransDef);
```

Этот новый тип `Simpl` может теперь использоваться с любой функцией транспортировки при упаковке или распаковке данных.

Приложению-адресату нет нужды беспокоиться об эндиан-ориентации источника. Когда адресат распаковывает данные этого типа, транспортный механизм автоматически корректирует эндиан-ориентацию, используя эндиан-определение в зарегистрированном транспортном типе. Это весьма выгодно в мультиплатформенном сетевом окружении. Если транспортный механизм используется для записи двоичных конфигурационных файлов, одни и те же файлы могут использоваться приложениями независимо от эндиан-ориентации машин, на которых они выполняются.

Более сложная структура

Вам часто надо транспортировать данные более сложных типов, которые ссылаются на внешние по отношению к себе данные (члены типа указателей). При выполнении операций упаковки и распаковки такие члены требуют специальную обработку. Для того, чтобы эти члены были обработаны надлежащим образом, они должны быть описаны в члене *fixup* элемента транспортного реестра.

Вот более сложная структура:

```
typedef struct simp2 {
    /* Скаляр и ссылка на скалярный массив */
    int num_ref_vals;
    int *ref_vals;

    /* Скалярный массив */
    int nums_10[10];

    /* Скалярный массив (не эндиан-чувствительный) */
    char first_name[10];

    /* Ссылка на строку */
    char *last_name2;

    /* Скалярный массив */
    short vals_5[5];

    /* Член зарегистрированного типа */
    Simpl_t simpl_instance;

    /* Массив членов зарегистрированного типа */
    Simpl_t simpl_array[4];

    /* Ссылка на зарегистрированный тип */
    Simpl_t *simpl_reference;

    /* Скаляр и ссылка на массив зарегистрированных типов */
    int num_simps;
    Simpl_t *ref_simpl_array;

    /* Скаляр и ссылка на массив ссылок зарегистрированных типов */
    int num_simpl_refs;
    Simpl_t **ref_simpl_ref_array;

    /* Два скаляра и ссылка на данные произвольного объема */
    short bm_height;
    int bm_bpl;
    char *bitmap;

    /* Нечто, что мы не хотим паковать, но хотим очистить после распаковки */
    char *dont_pack_this;
} Simp2_t;
```

Список ссылок на очистку

Вот список *clear_refs* для этой структуры:

```
static const int unsigned Simp2ClearRefs[] = {
    offsetof( Simp2_t, dont_pack_this ),
    0 /* Конец списка ссылок на очистку */
};
```

Эндиан-список

Вот эндиан-список для этой структуры:

```
static const int unsigned Simp2Endians[] = {
    Tr_ENDIAN( Simp2_t, num_ref_vals ),
    Tr_ENDIAN_REF( Simp2_t, ref_vals ),
};
```

```
Tr_ENDIAN_ARRAY( Simp2_t, nums_10 ),
Tr_ENDIAN_ARRAY( Simp2_t, vals_5 ),
0 /* Конец эндиан-списка */
};
```

Вот полный список эндиан-декларации для каждого типа членов:

Скаляр (char)	Отсутствует
Скаляр (short)	Tr_ENDIAN(<i>type</i> , <i>member</i>)
Скаляр (int)	Tr_ENDIAN(<i>type</i> , <i>member</i>)
Скалярный массив (char)	Отсутствует
Скалярный массив (short)	Tr_ENDIAN_ARRAY(<i>type</i> , <i>member</i>)
Скалярный массив (int)	Tr_ENDIAN_ARRAY(<i>type</i> , <i>member</i>)
Ссылка (char)	Отсутствует
Ссылка (short)	Tr_ENDIAN_REF(<i>type</i> , <i>member</i>)
Ссылка (int)	Tr_ENDIAN_REF(<i>type</i> , <i>member</i>)
Ссылка (массив char)	Отсутствует
Ссылка (массив short)	Tr_ENDIAN_REF(<i>type</i> , <i>member</i>)
Ссылка (массив int)	Tr_ENDIAN_REF(<i>type</i> , <i>member</i>)
Простая структура	Список каждого эндиан-чувствительного члена структуры членов
Зарегистрированный тип	Отсутствует
Ссылка на зарегистрированный тип	Отсутствует

Например, для структуры `Sample_t`:

```
int i;                Tr_ENDIAN( Sample_t, i )
int array[7];        Tr_ENDIAN_ARRAY( Sample_t, array )
short *short_nums;   Tr_ENDIAN_REF( Sample_t, short_nums )
int *long_nums;      Tr_ENDIAN_REF( Sample_t, long_nums )
struct my_simp ms;    Tr_ENDIAN( Sample_t, ms.width ), Tr_ENDIAN(
Sample_t, ms.height )
```

Fixup-список

Структура `Simp2_t`, приведенная выше, включает несколько членов, ссылающихся на данные вне этой структуры. Эти элементы требуют наличия членов типа `PhTransportFixupRec_t` в списке *fixup*, чтобы указать транспортному механизму, как получить данные:

```
static const PhTransportFixupRec_t
Simp2Fixups[] = {
    Tr_REF_ARRAY( Simp2_t, ref_vals, Tr_FETCH( Simp2_t, num_ref_vals ) ),
    Tr_STRING( Simp2_t, name2 ),
    Tr_TYPE( Simp2_t, simpl_instance ),
    Tr_TYPE_ARRAY( Simp2_t, simpl_array ),
    Tr_REF_TYPE( Simp2_t, simpl_reference ),
    Tr_REF_TYPE_ARRAY(
        Simp2_t, ref_simpl_array,
        Tr_FETCH( Simp2_t, num_simps ) ),
    Tr_REF_TYPE_REF_ARRAY(
        Simp2_t, ref_simpl_ref_array,
        Tr_FETCH( Simp2_t, num_simp_refs ) ),
    Tr_ALLOC(
        Simp2_t, bitmap,
        Tr_FETCH( Simp2_t, bm_bpl ), '*',
        Tr_FETCH( Simp2_t, bm_height ) )
};
```

При определении элемента в *fixup* Вам может понадобиться использовать информацию, находящуюся в структуре, элемент которой Вы определяете. В этой ситуации используйте декларацию:

`Tr_FETCH(type, member)` Эта декларация приводит к тому, что значение задаваемого члена будет использоваться во время исполнения – когда данные будут паковаться или распаковываться. Кроме того, любые члены, определённые через другие зарегистрированные типы, автоматически являются эндиан-корректными, используя эндиан-определение из элемента транспортного реестра.

Вот полный список деклараций адресной привязки:

Scalar	Скаляр. Отсутствует
Scalar Array	Массив скаляров. Отсутствует
Reference (string)	Ссылка (строка). <code>Tr_STRING(<i>type, member</i>)</code>
Reference (scalar array)	Ссылка (массив скаляров). <code>Tr_REF_ARRAY(<i>type, member, number_of_elements</i>)</code>
Registered type	Зарегистрированный тип. <code>Tr_TYPE(<i>type, member, type_name</i>)</code>
Registered type array	Массив зарегистрированных типов. <code>Tr_TYPE_ARRAY(<i>type, member, type_name</i>)</code>
Reference (registered type)	Ссылка (зарегистрированный тип). <code>Tr_REF_TYPE(<i>type, member, type_name</i>)</code>
Reference (registered type array)	Ссылка (массив зарегистрированных типов). <code>Tr_REF_TYPE_ARRAY(<i>type, member, num_elements, type_name</i>)</code>
Reference (registered type reference array)	Ссылка (массив ссылок на зарегистрированные типы). <code>Tr_REF_TYPEREF_ARRAY(<i>type, member, num_elements, type_name</i>)</code>

Вот несколько примеров членов и их деклараций адресной привязки:

```
char *name;
Tr_STRING( Sample_t, name )

int num_nums;
int *int_array;
Tr_REF_ARRAY( Sample_t, int_array, Tr_FETCH( Sample_t, num_nums ) )

или, если член известен:
Tr_REF_ARRAY( Sample_t, int_array, 7 )

Simpl_t simple_instance
Tr_TYPE( Sample_t, simple_instance, "simpl" )

или как один экземпляр, если это просто массив:
Tr_TYPE_ARRAY( Sample_t, simple_instance, "simpl" )

Simpl_t simple_array[5]
Tr_TYPE_ARRAY( Sample_t, simple_array, "simpl" )

Simpl_t *simpl_ref
Tr_REF_TYPE( Sample_t, simpl_ref, "simpl" )

short num_simpls;
Simpl_t *simpl_ref
Tr_REF_TYPE_ARRAY( Sample_t, simpl_ref, Tr_FETCH( Sample_t, num_simpls ), "simpl" )

short num_simpls;
Simpl_t **simpl_ref;
Tr_REF_TYPE_REF_ARRAY( Sample_t, simpl_ref,
                       Tr_FETCH( Sample_t, num_simpls ), "simpl" )
```

Элемент реестра

Наконец, вот элемент реестра для `Simp2_t`:

```
static const PhTransportRegEntry_t
Simp2TransDef = {
    "simp2",
    Ph_PACK_STRUCT,
    sizeof( Simp2_t ),
    sizeof(Simp2Fixups)/sizeof(Simp2Fixups[0]),
    &Simp2Fixups,
    &Simp2Endians,
    &Simp2ClearRefs
};
```

Транспортные функции

В этом разделе описываются низкоуровневые функции и типы данных, относящиеся к механизму транспортировки. Некоторые функции вызываются приложением, которое является источником данных. Некоторые функции вызываются приложением, которое является источником данных, некоторые – адресатом, некоторые – и тем и другим.

Оба приложения

Оба приложения используют:

<code>PhTransportRegEntry_t</code>	Структура данных, описывающая транспортируемые данные
<code>PhRegisterTransportType()</code>	Добавляет новый транспортный тип в реестр транспортировки
<code>PhFindTransportType()</code>	Отыскивает транспортный тип в реестре транспортировки

Приложение-источник

Приложение-источник использует это, примерно в таком порядке:

<code>PhTransportCtrl_t</code>	Структура управления для механизма транспортировки Photon'a
<code>PhCreateTransportCtrl()</code>	Выделяет память под структуру <code>PhCreateTransportCtrl()</code>
<code>PhTransportType()</code>	Упаковывает данные в структуру <code>PhTransportCtrl_t</code>
<code>PhTransportFindLink()</code>	Отыскивает связанный список данных транспортировки для некоторых специфических данных
<code>PhTransportLink_t</code>	Элемент связанного списка транспортных данных
<code>PhLinkTransportData()</code>	Добавляет транспортные данные в связанный список
<code>PhGetNextInlineData()</code>	Получает данные из следующего элемента связанного списка транспортных данных
<code>PhGetTransportVectors()</code>	Строит вектор ввода/вывода транспортируемых данных
<code>PhFreeTransportType()</code>	Освобождает данные, связанные с элементом транспортного реестра
<code>PhReleaseTransportCtrl()</code>	Освобождает структуру <code>PhTransportCtrl_t</code>

Это низкоуровневые функции, которые Вам, вероятно, никогда не понадобятся вызывать непосредственно:

<code>PhAllocPackType()</code>	Выделяет буфер и упаковывает в него транспортные данные
<code>PhPackEntry()</code>	Упаковывает транспортные данные, установленные элементом транспортного реестра
<code>PhPackType()</code>	Упаковывает транспортные данные, установленные типом данных

Приложение-адресат

Приложение-адресат использует следующее, примерно в таком порядке:

PhGetAllTransportHdrs()	Извлекает все заголовки из буфера упакованных транспортных данных
PhGetTransportHdr()	Извлекает заголовок из буфера упакованных транспортных данных
PhGetNextTransportHdr()	Получает следующий заголовок из буфера упакованных транспортных данных
PhLocateTransHdr()	Ищет специфические данные в связанном списке транспортных заголовков
PhMallocUnpack()	Распаковывает транспортные данные, используя самостоятельно написанную функцию выделения памяти
PhUnpack()	Распаковывает транспортные данные
PhUnlinkTransportHdr()	Удаляет элемент из связанного списка транспортных заголовков
PhReleaseTransportHdrs()	Освобождает связанный список заголовков упакованных транспортных данных

Глава 22. Регионы

В Photon'e все приложения состоят из одного или более прямоугольников, называемых *регионами* (*regions*), находящихся в некоем абстрактном трёхмерном *пространстве событий*. Регионам назначены идентификационные номера типа PhRid_t.

В этой главе обсуждается:

- Координатное пространство Photon'a
- Координаты региона
- Регионы и отсечение событий
- Месторасположение и иерархия
- Использование регионов
- Системная информация

☞ Вы можете использовать утилиту `phview`, чтобы посмотреть, какие регионы существуют на Вашей машине. Для более подробной информации см. "Справочник утилит".

Координатное пространство Photon'a

Координатное пространство Photon'a выглядит так:

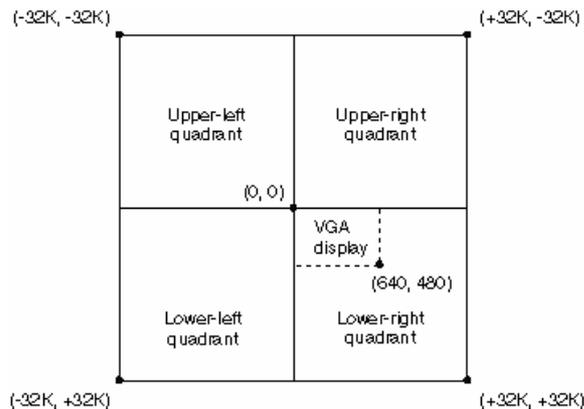


Рис. 22-1. Координатное пространство Photon'a

☞ В отличие от обычной декартовой разметки, квадрантом (+, +) является нижний правый.

Корневой регион имеет те же размеры, что и координатное пространство в целом. Как правило, графические драйверы отображают экран дисплея в местоположение, показанное на вышеприведенном рисунке и размещают начало координат Photon'a в левом верхнем углу экрана дисплея. (Графические драйверы приравнивают единицу координаты Photon'a единице значения пикселей экрана Вашего дисплея).

Координаты региона

Начало координат региона

Когда какое-то приложение задаёт координаты внутри данного региона, они являются относительными – от начала координат региона. Приложение задаёт это начало, когда оно открывает регион.

Начальные размеры и расположение

Начальные размеры региона (т.е. аргумент *rect* функции `PhRegionOpen()`) являются относительными от его начала координат. Эти размеры управляют диапазоном координат, которые приложение может использовать внутри региона. Давайте рассмотрим несколько примеров, чтобы осознать взаимосвязь между началом координат региона и начальными координатами его прямоугольника. Эти примеры демонстрируют, как открытые регионы размещаются относительно корневого региона, имеющего начало координат в центре координатного пространства Photon'a.

Начало координат в (0,0) и начальный прямоугольник в (0,0)

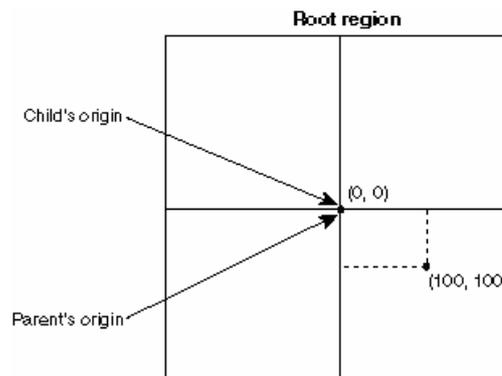
По умолчанию приложение использует для регионов следующий подход (разновидности регионов описаны в разделе "Оконный менеджер Photon'a" приложения "Архитектура Photon'a").

Координаты:

Начальные = (0,0)

Верхний левый угол начального прямоугольника = (0,0)

Нижний правый угол начального прямоугольника = (100,100)



Начало координат в (0,0) и начальный прямоугольник не в (0,0)

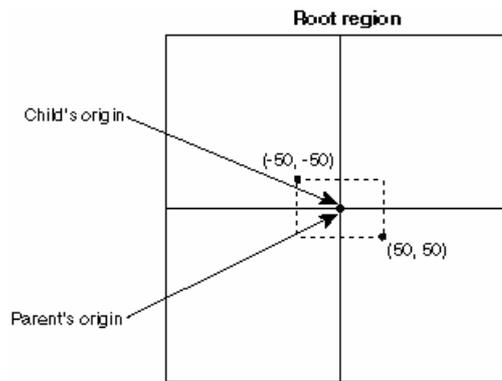
Следующий пример показывает подход, обычно используемый для регионов, которые заполняют целиком координатное пространство, например, для региона устройств и региона рабочей области (workspace), верхний левый угол имеет координаты (-32768, -32768) и нижний правый – (32767, 32767).

Координаты:

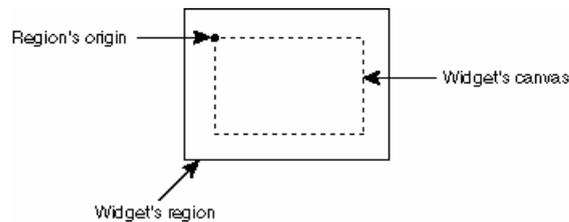
Начальные = (0,0)

Верхний левый угол начального прямоугольника = (-50,-50)

Нижний правый угол начального прямоугольника = (50,50)



Многие виджеты создают регионы, верхние левые углы которых имеют отрицательные координаты, так что начало координат холста виджета находится в $(0,0)$:



Начало координат не в $(0,0)$ и начальный прямоугольник не в $(0,0)$

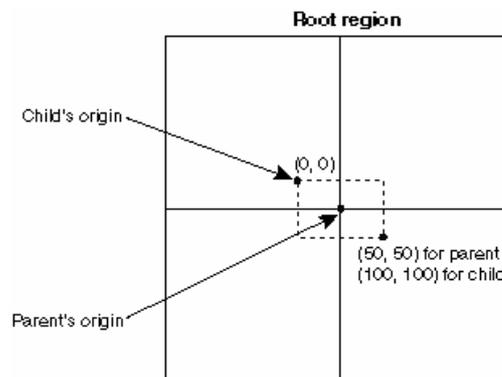
Следующий пример показывает, как начало координат потомка может отличаться от начала координат родителя.

Координаты:

Начальные: $= (-50, -50)$

Верхний левый угол начального прямоугольника $= (0,0)$

Нижний правый угол начального прямоугольника $= (100, 100)$



О регионах потомка

Начало координат региона потомка задаётся относительно начала координат родителя. Таким образом, когда регион перемещается, все его потомки автоматически перемещаются вместе с ним. И также, когда регион уничтожается, уничтожаются его потомки.

- ☞ Если Вы хотите сделать регион большим по размеру, чем какой-либо другой регион Вашего приложения, сделайте его потомком корневого региона, вызвав функцию `PhRegionOpen()` или `PhRegionChange()`, задав `Ph_ROOT_RID` как родительский.

Регионы и отсечение событий

Регион может генерировать или получать события только там, где он перекрывается со своим родителем. Например, на следующем рисунке:

- Потомок 1 может генерировать или получать события во всём своём регионе
- Потомок 2 может генерировать или получать события только в той малой серой зоне, которая перекрывается с регионом родителя.

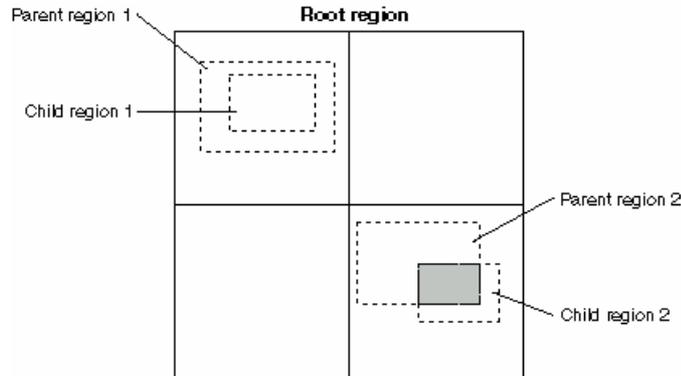


Рис. 22-2. Регионы и отсечение событий

На основании этого свойства регионов, любая часть региона, которая не перекрывается со своими родителями, в действительности является невидимой.

Месторасположение и иерархия

Иерархия регионов:

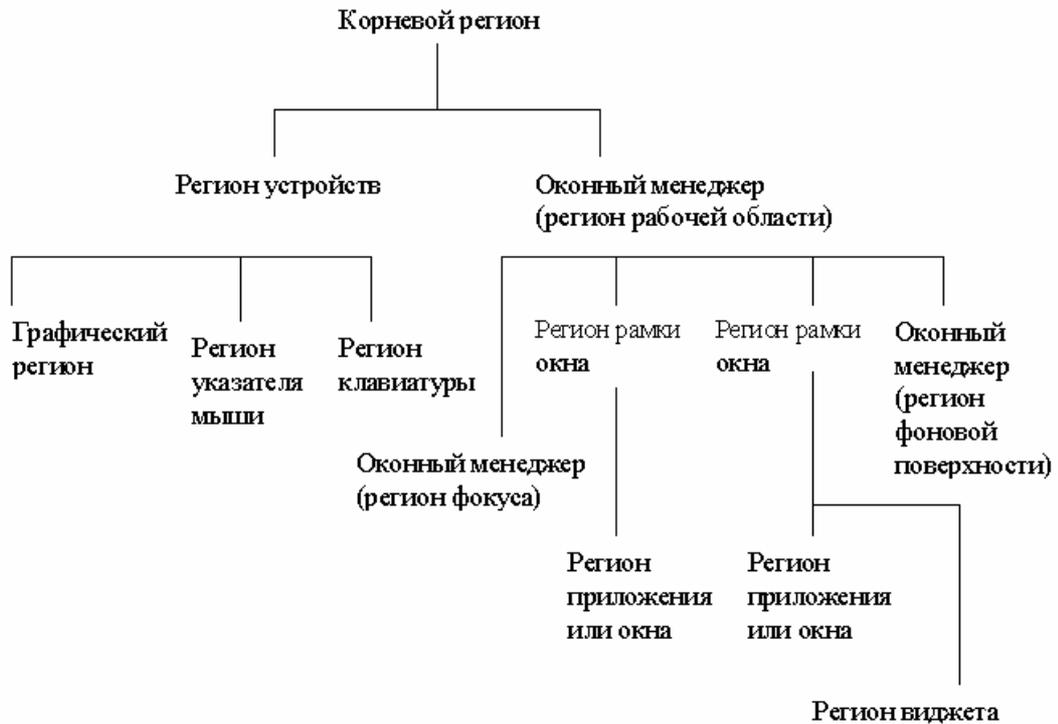
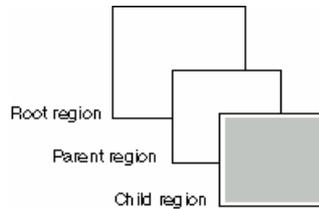


Рис. 22-3. Иерархия регионов типичной системы Photon'a

В Photon'e каждый регион имеет родительский регион. Эта связь родитель-потомок приводит к появлению иерархии регионов с корневым регионом в вершине. Следующая диаграмма показывает иерархию типичной системы Photon'a:

Родительский регион

Менеджер Photon'a всегда размещает регионы потомков впереди (т.е. в сторону пользователя) своих родителей:

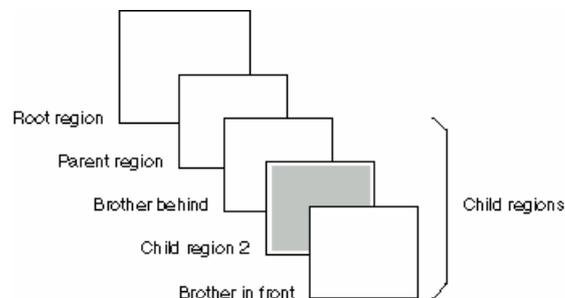


☞ При открытии региона приложение определяет родителя региона. Если приложение открывает регион, не задавая его родителя, родитель региона устанавливается по умолчанию – базовые регионы становятся потомками корневого региона и окна становятся потомками фонового региона оконного менеджера.

Братские регионы

Кроме родителя, регион может иметь "братьев", т.е. другие регионы, имеющие того же родителя. Регион знает только о своих двух братьях – о том, который непосредственно перед ним, и том, который непосредственно за ним.

На следующем рисунке показан родитель с тремя потомками, и взаимосвязи, которые имеются между регионом потомка 2 с его братьями:



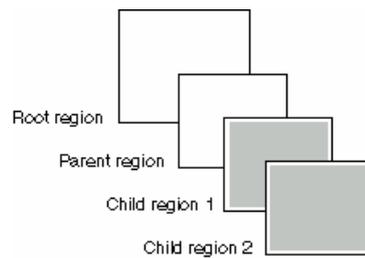
Когда приложение открывает регион (напр., регион потомка 2 на вышеприведенном рисунке), оно может либо не задавать ничего, либо задать одного, либо двух непосредственных братьев. В зависимости от того, как приложение задало этих братьев, новый регион может быть размещён в соответствии с принимаемыми по умолчанию правилами (см. ниже) или в определённом месторасположении.

☞ Если приложение открывает регион, задав обоих братьев, и результатом этого является неоднозначные требования по месторасположению, требования отвергаются.

Месторасположение по умолчанию

Если приложение открывает регион, не задавая братьев, менеджер Photon'a размещает этот регион, используя принимаемые по умолчанию правила месторасположения. В большинстве случаев эти правила приводят к тому, что вновь открытый регион располагается впереди своего самого переднего брата, который становится "братом сзади" для нового региона. (Чтобы использовать другие правила месторасположения, Вы можете задать флаг `Ph_FORCE_FRONT`).

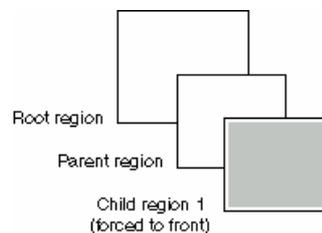
Например, на следующем рисунке регион потомка 1 является самым передним регионом:



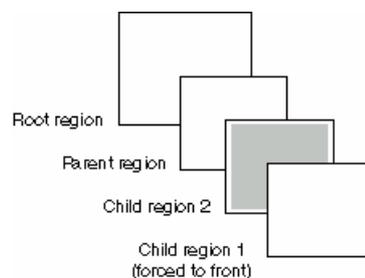
Когда приложение открывает регион потомка 2 с принимаемым по умолчанию месторасположением (следующий рисунок), регион 2 размещается впереди региона 1. Регион 1 становится братом сзади региона 2. Регион 2 становится братом спереди региона 1.

Флаг Ph_FORCE_FRONT

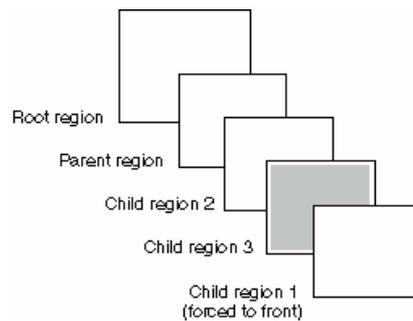
Приложение использует флаг `Ph_FORCE_FRONT`, когда оно хочет, чтобы регион оставался впереди всех последующих братьев, зависящих от принимаемого менеджера Photon'a по умолчанию месторасположением. Как обсуждалось ранее, когда регион открыт с принимаемым по умолчанию месторасположением, он размещается перед своим самым передним братом. Но если какой-либо брат имеет установленный флаг `Ph_FORCE_FRONT`, новый регион размещается *позади* самого заднего брата, у которого установлен флаг `Ph_FORCE_FRONT`. Например, давайте посмотрим, что случится в следующем примере, если регион потомка 1 имеет установленный флаг `Ph_FORCE_FRONT`:



Когда открывается регион потомка 2 с принимаемым по умолчанию месторасположением (следующая диаграмма), он размещается позади региона 1, и регион 1 становится "братом впереди". Поскольку регион 2 был размещён с использованием принимаемых по умолчанию правил, он не наследует установку `Ph_FORCE_FRONT` региона 1:



Затем, если открывается регион 3 с принимаемым по умолчанию месторасположением, он размещается следующим образом:



- ☞ Приложение может установить флаг `Ph_FORCE_FRONT` при открытии региона (или позже), изменив флаги региона. Состояние этих флагов не оказывает влияние на то, как размещается сам регион – они влияют на то, как размещаются последующие братья, если эти братья открываются с использованием принимаемых по умолчанию правил месторасположения. То есть состояние `Ph_FORCE_FRONT` существующих братьев не оказывает влияние на месторасположение нового региона, если он открывается с заданными связями с братом. См. следующий раздел – "Задаваемое месторасположение".

Помните, что флаг `Ph_FORCE_FRONT` оказывает влияние только на местоположение регионов своих братьев – регион потомка всегда становится впереди его родителя.

Задаваемое месторасположение

В отличие от месторасположения, принимаемого по умолчанию, если при открытии региона определяется какой-либо из братьев, то это определение управляет месторасположением нового региона. Мы называем это *задаваемым месторасположением*.

Если определяется брат позади, то новооткрытый регион автоматически размещается перед этим братом.

Если определяется брат впереди, то новооткрытый регион автоматически размещается позади этого брата.

- ☞ Установка `Ph_FORCE_FRONT` задаваемого брата наследуется новым регионом. Если приложение открывает регион, задавая обоих братьев, и это приводит к неоднозначным требованиям месторасположения, то открытие не выполняется.

Использование регионов

Открытие региона

Чтобы открыть регион, создайте виджет `PtRegion`. Виджет `PtRegion` не включён в палитру PhAB'a; для его реализации:

- Вызовите в Вашем приложении функцию `PtCreateWidget()`
или
- Создайте модуль окна, выберите его и используйте пункт "Change Class" в меню "Edit" PhAB'a, чтобы превратить окно в `PtRegion`. Для более полной информации см. раздел "Изменение класса виджета" в главе "Создание виджетов в PhAB".

Более полную информацию по виджету `PtRegion` см. в "Справочнике виджетов".

Размещение регионов

В то время как регион всегда размещается перед своим родителем, местоположение его относительно своих братьев является гибким. См. раздел "Месторасположение и иерархия" для получения более полной информации о местоположении "по умолчанию" и "задаваемом".

Структура `PhRegion_t` (см. "Справочник библиотечных функций Photon'a") содержит следующие члены. Они указывают на взаимосвязь региона со своими братьями:

- `bro_in_front` – указывает на брата непосредственно впереди.
- `bro_behind` – указывает на брата непосредственно сзади.

Для получения этой информации Вы можете использовать функцию `PhRegionQuery()`.

Изменение месторасположения региона

Приложение может задать месторасположение региона при его открытии или оно может изменить месторасположение позже. Чтобы изменить месторасположение региона, приложение должно изменить взаимосвязь между регионом и семейством региона. Приложение делает это, выполняя что-либо или всё из нижеследующего:

- устанавливает члены `parent`, `bro_front` и `bro_behind` в структуре `PhRegion_t`;
- устанавливает соответствующие биты в `fields` для указания на то, какие члены являются действительными (только те члены, которые помечены как валидные, будут оказывать действие);
- вызывает функцию `PhRegionChange()`.

☞ Поскольку приложение может быть уверенным в позиции только принадлежащих ему регионов, оно не в состоянии изменить позицию каких-либо других регионов. В противном случае, в то время, когда приложение выполняет запрос на изменение позиции региона, ему не принадлежащего, доставленная функцией `PhRegionQuery()` может не отражать текущую позицию этого региона. То есть запрос на изменение месторасположения региона может не иметь результата, на который рассчитывает приложение.

Изменение родителя

Вы можете изменить родительский регион таким способом:

- Если регион имеет родительский виджет, вызовите функцию `PtReparentWidget()`, чтобы сделать регион потомком другого виджета. Не переназначайте родителя региона напрямую.
- Задайте родителя в члене `parent` структуры `PhRegion_t` потомка. Регион потомка становится самым передним среди регионов-потомков родителя.
- Задайте потомка другого родителя как брата данного региона. Это делает регион потомком этого другого родителя, но позволяет Вам задать, где регион потомка размещается в иерархии регионов родителя.

В заголовочном файле `<photon/PhT.h>` определены следующие константы:

- `Ph_DEV_RID` – идентификатор региона устройств
- `Ph_ROOT_RID` – идентификатор корневого региона

Задаваемые братья

Если Вы устанавливаете:	То:
<code>bro_behind</code>	Регион, указанный в члене <code>rid</code> структуры <code>PhRegion_t</code> , перемещается перед регионом <code>bro_behind</code>
<code>bro_in_front</code>	Регион, указанный в члене <code>rid</code> структуры <code>PhRegion_t</code> , перемещается за регион <code>bro_in_front</code>

Как обсуждалось в разделе "Изменение родителя", регион наследует родителя какого-либо из задаваемых братьев, которые являются потомками другого родителя.

Системная информация

Вы можете получить о Вашей системе следующую информацию:

- версию Вашего сервера Photon'a
- оценку пропускной способности связи между Вашим окном и сервером Photon'a
- информацию о регионах, перекрывающих Ваше окно:
 - графические регионы
 - регионы клавиатуры
 - регионы указателя мыши
 - регионы группы ввода

☞ Вы не получите информацию о каждом регионе. Вместо этого Вы получаете минимальное значение каждого типа информации. Например, если несколько регионов графического драйвера, перекрывающих Ваше окно, имеют различную пропускную способность, выдаваемая пропускная способность является минимальной из них.

Имеются две функции, которые Вы можете использовать для получения системной информации:

`PhQuerySystemInfo()` Получение информации о данном регионе

`PtQuerySystemInfo()` Получение информации о виджете (обычно окно)

`PhQuerySystemInfo()` отсылает сообщение серверу каждый раз, когда Вы её вызываете.

Функция `PtQuerySystemInfo()` вызывает функцию `PhQuerySystemInfo()`, но буферизует информацию. Когда регион, перекрывающий Ваш виджет, изменяется (например, он перемещается), буфер помечается как недействительный. В следующий раз, когда Вы вызовете `PtQuerySystemInfo()`, она вновь вызовет функцию `PhQuerySystemInfo()`. Используя буфер всегда, когда это возможно, функция `PtQuerySystemInfo()` удерживает минимальным количество сообщений.

Обе функции – и `PtQuerySystemInfo()`, и `PhQuerySystemInfo()` – заполняют структуру типа `PhSysInfo_t`, которую выделяет в памяти Ваше приложение. Более полная информация – в "Справочнике библиотечных функций Photon'a".

Особый интерес представляет одна область – графическая полоса пропускания в `gfx.bandwidth`. Это значение может быть использовано для модификации поведения интерфейса, основанного на скорости связи. Например, простое изменение состояния может заменить изошрённую мультипликацию, если полоса пропускания равна `Ph_BAUD_SLOW` или ниже. Также хорошей идеей является выполнение проверки того, может ли совместно используемая память применяться для рисования; флаг `Ph_GCAP_SHMEM` в `gfx.capabilities` установлен, если все графические драйверы поддерживают функции семейства `...mx()` и все они запущены на Вашем узле.

Глава 23. События

Взаимодействия между приложениями, пользователями и сервером Photon'a представлено через структуры данных, называемых событиями. В этой главе обсуждается:

- События мыши [*"Pointer events" – в общем случае, конечно, не мыши, а курсора манипулятора. Но для лучшего понимания пишу – мышь. Ибо как иначе перевести "press the pointer button"??? – Прим. пер.]*
- Генерация события
- Координаты события
- Обработчики события – необработанные и отфильтрованные ответные реакции
- Накопление событий
- Сжатие событий
- Перетаскивание

Информация по событию хранится в структуре типа `PhEvent_t`; см. "Справочник библиотечных функций Photon'a".

События мыши

Чаще всего при обработке того, что пользователь делает, когда указывает на виджет, Вы будете использовать ответные реакции этого виджета.

Нажатие кнопки

Когда Вы нажимаете на кнопку мыши, Photon генерирует событие `Ph_EV_BUT_PRESS` для виджета, имеющего в этот момент фокус.

Отпускание кнопки

Когда Вы отпускаете нажатую кнопку, Photon генерирует **два** события `Ph_EV_BUT_RELEASE`:

- Одно событие подтипа `Ph_EV_RELEASE_REAL`
- Одно – подтипа `Ph_EV_RELEASE_PHANTOM`

Когда Вы отпускаете кнопку мыши, реальное отпускание попадает на указатель мыши; фантомное отпускание *всегда* проходит на тот же регион (и позицию), который получил нажатие кнопки.

Другими словами, если Ваш виджет увидел нажатие, он также увидит и фантомное отпускание. И в зависимости от того, где был указатель мыши, вы можете получить, а можете и не получить реальное отпускание. Если Ваш виджет получил и реальное и фантомное отпускание, реальное всегда приходит первым.

Несколько щелчков

Нажимаете ли Вы или отпускаете кнопку мыши, событие включает в себя счётчик щелчков. Как Ваше приложение определяет, что Вы щёлкнули, а не выполнили двойной щелчок?

В данных по событию имеется счётчик щелчков, связанный с событиями `Ph_EV_BUT_PRESS` и `Ph_EV_BUT_RELEASE`; чтобы получить эти данные, вызовите функцию `PhGetData()`. Данными для этих событий являются структуры типа `PhPointerEvent_t` (см. подробнее в "Справочнике библиотечных функций Photon'a"); её член `click_count` даёт количество Ваших щелчков кнопкой мыши.

Если Вы продолжаете щёлкать достаточно быстро, не двигая мышь, счётчик продолжает возрастать. Если Вы перемещаете мышь или на время прекращаете щёлкать мышью, счётчик сбрасывается и Photon генерирует событие `Ph_EV_BUT_RELEASE` с подтипом `Ph_EV_RELEASE_ENDCLICK`.

Иными словами, первый щелчок генерирует событие `Ph_EV_BUT_PRESS` и пару событий `Ph_EV_BUT_RELEASE` (одно реальное REAL и одно фантомное PHANTOM) со счётчиком `click_count`, установленным в 1. Затем, в зависимости от того, щёлкнул ли пользователь достаточно быстро вновь или нет, Вы получите либо

- событие `Ph_EV_BUT_PRESS` и пару событий `Ph_EV_BUT_RELEASE` со счётчиком `click_count`, установленным в 2;
либо
- событие `Ph_EV_BUT_RELEASE` с подтипом `Ph_EV_RELEASE_ENDCLICK`.

После второго щелчка Вы либо получаете третий, либо ENDCLICK, и так далее. Но в конечном счёте Вы получаете ENDCLICK – и в следующий раз, когда пользователь щёлкнет, счётчик щелчков вновь равен 1.

Клавиши-модификаторы

Если Вам надо определить, какие клавиши были в событии мыши, вызовите функцию `PhGetData()`, чтобы получить данные по событию, которые включают события `Ph_EV_BUT_PRESS` и `Ph_EV_BUT_RELEASE`. Данными для этих событий является структура типа `PhPointerEvent_t` (описанная в "Справочнике библиотечных функций Photon'a"); проверьте её член `key_mods`, чтобы определить клавиши-модификаторы, которые были нажаты.

Например, эта ответная реакция `Pt_CB_ACTIVATE` предоставляет список клавиш-модификаторов, которые были нажаты, когда была отпущена кнопка мыши:

```

/* Стандартные хедеры */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

/* Инструментальные хедеры */
#include <Ph.h>
#include <Pt.h>
#include <Ap.h>

/* Локальные хедеры */
#include "abimport.h"
#include "proto.h"

int check_keys( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo ) {
    PhPointerEvent_t *event_data;

    /* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    widget = widget, apinfo = apinfo, cbinfo = cbinfo;

    if (cbinfo->event->type != Ph_EV_BUT_RELEASE) {
        printf ("Это не событие Ph_EV_BUT_RELEASE\n");
    }
    else {
        printf ("Это событие Ph_EV_BUT_RELEASE\n");

        event_data = (PhPointerEvent_t *) PhGetData (cbinfo->event);

        if (event_data->key_mods & Pk_KM_Shift ) printf (" Shift\n");

        if (event_data->key_mods & Pk_KM_Ctrl ) printf (" Ctrl\n");

        if (event_data->key_mods & Pk_KM_Alt ) printf (" Alt\n");

    }
    return( Pt_CONTINUE );
}

```

Генерирование событий

В этом разделе описывается использование функции `PhEmit()`, а также

- нацеливание на определённые регионы
- нацеливание на определённые виджеты
- события, генерируемые клавиатурными клавишами

Основным способом генерирования событий в Вашем приложении является вызов функции `PhEmit()`:

```
int PhEmit(PhEvent_t .event,
           PhRect_t   .rects,
           void       .data);
```

Её аргументами являются:

event Указатель на структуру `PhEvent_t`. Приложение, генерирующее событие, должно установить следующие члены:

- *type* – тип события.
- *subtype* – подтип события (если необходимо).
- *flags* – модификаторы события (напр., направление).
- *emitter* – структура типа `PhEventRegion_t`; Вам необходимо установить по меньшей мере идентификатор региона, генерирующего событие.
- *translation* – при генерировании события обычно устанавливается в (0,0).
- *num_rects* – количество прямоугольников в аргументе *rects* функции. Если Вы устанавливаете значение *num_rects* в 0, Вы также должны передать в качестве *rects* `NULL`.
- *event->collector.rid* – если Вы устанавливаете идентификатор накопителя в 0, событие ставится в очередь для каждого должным образом чувствительного региона, интересующегося событием.

Если вы устанавливаете идентификатор накопителя равным идентификатору какого-то региона, только этот регион замечает событие.

Менеджер `Photon'a`, после того, как поставил копию события в очередь к приложению, устанавливает в структуре события следующие члены:

- *timestamp* – время, когда событие было сгенерировано (в миллисекундах);
- *collector* – структура типа `PhEventRegion_t`, которая включает идентификатор региона-накопителя.

rects Массив структур `PhRect_t` (см. "Справочник библиотечных функций `Photon'a`"), указывающих на набор начальных прямоугольников события. Если этот аргумент является `NULL`'ом, набор состоит из единственного прямоугольника, соответствующего генерирующему региону.

data Данные, действительные для того типа события, которое было сгенерировано. Каждый тип событий имеет свой собственный тип данных, как описано в описании структуры `PhEvent_t` в "Справочнике библиотечных функций `Photon'a`".

Если специфические для события данные не находятся в примыкающей области памяти, функция `PhEmitmx()` может Вам показаться более полезной, нежели `PhEmit()`:

```
int PhEmit(PhEvent_t *event,
           PhRect_t  *rects,
           int       mxparts,
           struct_mxfer_entry *mx);
```

Кодами возврата функций `PhEmit()` и `PhEmitmx()` являются:

Неотрицательное значение	Успешное завершение
-1	Произошла ошибка; значение ошибки установлено в <i>errno</i>

Нацеливание на определённые регионы

Иногда приложению необходимо нацелить событие непосредственно на определённый регион, без совершения путешествия события через всё пространство событий, до прибытия на этот регион. Вы можете использовать неисключительное событие (*inclusive event*) или направленное событие (*direct event*), чтобы обеспечить обнаружение события конкретными регионами.

Неисключительное событие

Для неисключительного события выполните следующее:

- Установите идентификатор генерирующего региона (т.е. *event->emitter.rid*) равным значению идентификатора целевого региона – это приведёт к тому, что событие будет сгенерировано автоматически *из* этого региона.
- установите флаг `Ph_EVENT_INCLUSIVE` в члене *flags* события – в результате менеджер Photon'a сгенерирует это событие в генерирующий регион перед тем, как запускать его в пространство событий.

Если Вы не хотите, чтобы неисключительно нацеленное событие продолжило прохождение через пространство событий, Вы должны сделать генерирующий регион *непрозрачным* для этого типа событий либо использовать вместо этого направленное событие.

Направленное событие

Для направленного события выполните следующее:

- Установите идентификатор генерирующего региона (т.е. *event->emitter.rid*) равным значению идентификатора региона Вашего приложения.
- Установите идентификатор региона-накопителя (т.е. *event->collector.rid*) равным значению идентификатора целевого региона.
- Установите флаг `Ph_EVENT_DIRECT` в члене *flag* события – это приведёт к тому, что менеджер Photon'a сгенерирует событие непосредственно из генерирующего региона в регион-накопитель.

Нацеливание на определённые виджеты

Если Вы хотите послать событие на определённый виджет, Вы можете вызвать функцию `PhEmit()`, как это описано выше, но Вам необходимо отследить массу мелочей, в том числе убедиться, что

- событие доставлено в надлежащее окно
- виджет ещё имеет фокус – перед Вашим событием в очередь могли поступить и другие события.

Проще вызвать функцию `PtSendEventToWidget()`. Эта функция отдаёт событие заданному виджету напрямую и без задержки. Это действует значительно более определённо и эффективно, чем в случае применения функции `PhEmit()`. Прототип этой функции такой:

```
int PtSendEventToWidget(PtWidget_t *widget,
                       PhEvent_t *event);
```

События, генерируемые клавиатурными клавишами

Иногда Вам может понадобиться в Вашем приложении эмулировать нажатие клавиши. В зависимости от того, чего именно Вы желаете достичь, Вы можете выбрать из нескольких способов генерирования событий клавиатуры:

- Генерировать событие `Ph_EVENT_KEY` из региона устройств:

```
event->emitter.rid = Ph_DEV_RID;
```

Набор прямоугольников должен состоять из одного пикселя – если Вы не используете менеджер окон, или, если Photon'овский менеджер окон установлен на использование фокуса курсора, позиция этого пикселя определит, какое окно откликнется на событие.

- Если Вы знаете регион, которому Вы хотите отослать событие, сгенерируйте событие Ph_EV_KEY непосредственно на этот регион:

```
event ->collector.rid = rid;
event ->flags |= Ph_EVENT_DIRECT;
```

Вот пример:

```
static void send_key( long key ) {
struct {
    PhEvent_t event;
    PhRect_t rect;
    PhKeyEvent_t pevent;
} new_event;

PhEvent_t event;
PhKeyEvent_t key_event;

PhRect_t rect;

rect.ul.x = rect.ul.y = 0;
rect.lr.x = rect.lr.y = 0;

memset( &event, 0, sizeof(event));
memset( &key_event, 0, sizeof(key_event) );

event.type = Ph_EV_KEY;
event.emitter.rid = Ph_DEV_RID;
event.num_rects = 1;
event.data_len = sizeof(key_event);
event.input_group = 1;

key_event.key_cap = key;
key_event.key_sym = key;

if ( isascii( key ) && isupper( key ) ) {
    key_event.key_mods = Pk_KM_Shift;
}

/* Генерирование нажатия клавиши */

key_event.key_flags = Pk_KF_Sym_Valid | Pk_KF_Cap_Valid | Pk_KF_Key_Down;
PhEmit( &event, &rect, &key_event );

/* Генерирование отпускания клавиши */

key_event.key_flags &= ~(Pk_KF_Key_Down | Pk_KF_Sym_Valid) ;
PhEmit( &event, &rect, &key_event );

return;
}
```

Координаты события

Когда генерируется событие, координаты его прямоугольника устанавливаются относительно начала координат генерирующего региона. Но когда событие принимается, его координаты становятся относительными к началу координат региона-накопителя.

Менеджер Photon'a обеспечивает это путём соответствующего пересчёта координат. Член *translation* в структуре PhEvent_t задаёт пересчёт между началами координат генерирующего региона и региона-накопителя.

Обработчики события – необработанные и отфильтрованные ответные реакции

Виджетный класс `PtWidget` предоставляет для обработки событий такие ответные реакции:

- `Pt_CB_FILTER` Вызывается *перед* тем, как событие обработано виджетом. Это позволит Вам осуществить на основании события действие до того, как виджет обнаружит событие. Это также даст Вам хорошую возможность принять решение о том, надо ли событие проигнорировать, снять или позволить виджету его обработать.
- `Pt_CB_RAW` Эти ответные реакции вызываются *после* того, как виджет обработал событие, даже если методы виджетного класса это событие поглощают.

Эти ответные реакции вызываются каждый раз, когда принимается событие `Photon'a`, совпадающее с маской событий (предоставленной приложением). Поскольку все классы виджетов библиотеки виджетов `Photon'a` являются потомками класса `PtWidget`, эти ответные реакции могут использоваться *любым* виджетом библиотеки виджетов `Photon'a`.

- ☞ Когда Вы прикрепляете к виджету необработанную или отфильтрованную ответную реакцию, библиотека виджета создаёт, если это необходимо, регион, который будет отлавливать для виджета заданные события. Это увеличивает количество регионов, которыми должен управлять менеджер `Photon'a`, и как результат, может приводить к понижению производительности. Из этих соображений используйте обработчики событий только тогда, когда Вам надо делать нечто, что не может быть выполнено с помощью стандартных ответных реакций виджета. Если Вы всё-таки используете обработчики событий, рассмотрите возможность использования их только в оконных виджетах, которые уже имеют регионы.

Каждый раз, когда поступает событие `Photon'a`, оно спускается по иерархии семейства виджета до тех пор, пока виджет его не поглотит. (Когда виджет обработал событие и исключил взаимодействие другого виджета с этим событием, говорят, что первый виджет *поглотил* событие).

В основном ответные реакции `Pt_CB_FILTER` вызываются при проходе вниз по иерархии, а ответные реакции `Pt_CB_RAW` – при проходе вверх. Каждый виджет обрабатывает событие подобным образом:

1. Ответные реакции `Pt_CB_FILTER` виджета вызываются, если тип события совпадает с маской ответной реакции. Код возврата ответной реакции указывает, что произошло с событием:

<code>Pt_CONSUME</code>	Событие поглощено, без обработки методами класса виджета.
<code>Pt_PROCESS</code>	Методам класса виджета было позволено обработать событие.
<code>Pt_IGNORE</code>	Событие проигнорировало виджет и всех его потокомков, как будто их и не существовало.

2. Если виджет чувствителен к событию и разрешена ответная связь `Pt_CB_FILTER`, метод виджетного класса обрабатывает событие. Метод класса может поглотить событие.
3. Если виджет поглотил событие, вызываются ответные реакции `Pt_CB_RAW` – если тип события совпадает с маской ответной реакции. Необработанные ответные реакции родителей виджета не вызываются.
4. Если виджет не поглотил событие, событие проходит на потомков виджета, если таковые имеются.

5. Если никакой виджет не поглотил событие, оно проходит обратно по иерархии семейства, и вызывается каждая ответная реакция Pt_CB_RAW виджета (если тип события совпадает с маской ответной реакции). Значение, возвращаемое ответной реакцией Pt_CB_RAW виджета, указывает, что произошло с событием:

Pt_CONSUME Событие поглотилось, и при прохождении вверх к родителю виджета никакие другие необработанные ответные реакции не вызывались
 Pt_CONTINUE Событие прошло вверх к родителю виджета

- ☞ Если виджет отключён (напр., в его флагах Pt_ARG_FLAGS выставлен флаг Pt_BLOCKED), необработанные и отфильтрованные ответные реакции не вызываются. Вместо них вызываются (если имеются) ответные реакции Pt_CB_BLOCKED виджета.

Давайте рассмотрим простенькое семейство виджетов, чтобы посмотреть, как это всё работает. Допустим, Вы имеете окно, содержащее панель, которая содержит кнопку. Вот что обычно происходит, когда Вы щёлкаете по кнопке:

1. Вызываются ответные реакции Pt_CB_FILTER окна, но событие не поглощается. Методы класса виджета тоже не поглощают событие.
2. Событие проходит к панели. Ни её ответные реакции Pt_CB_FILTER, ни её методы класса не поглощают событие.
3. Событие проходит к кнопке. Её ответные реакции Pt_CB_FILTER не поглощают событие, но это делают методы её класса; вызывается соответствующая ответная реакция (напр., Pt_CB_ACTIVATE).
4. Для события вызывается ответная реакция Pt_CB_RAW кнопки.
5. Ответные реакции Pt_CB_RAW панели и окна не вызываются, поскольку кнопка поглотила событие.

Если ответная реакция Pt_CB_FILTER панели указывает проигнорировать событие:

1. Окно обрабатывает событие, как и раньше
2. Ответная реакция Pt_CB_FILTER панели указывает проигнорировать событие, так что панель и все её потомки пропускаются.
3. Больше виджетов в семье нет, так что вызывается ответная реакция Pt_CB_RAW окна.

Более подробно о добавлении обработчиков событий смотри в:

- разделе "Обработчики событий – необработанные и отфильтрованные ответные реакции" в главе "Редактирование ресурсов и ответных реакций в PhAB";
- разделе "Обработчики событий" главы "Управление виджетами в программном коде приложения".

Накопление событий

Большинство приложений набирают события через вызов функции PtMainLoop(). Эта подпрограмма обрабатывает события Photon'a и поддерживает рабочие процедуры и обработку ввода. Если в Вашем приложении виджеты не используются, Вы можете накапливать события:

- *асинхронно*, вызывая функцию PhEventRead(). Перед тем, как в первый раз вызвать PhEventRead(), Вы должны вызвать функцию PhEventArm().
- *синхронно*, вызывая функцию PhEventNext(). Вы можете проверять наличие событий без блокировки, вызывая PhEventPeek().

Однако написание Вашей собственной функции главной петли не является тривиальной задачей; проще создать отдельный виджет (такой как PtRegion или PtWindow) и затем использовать PtMainLoop().

Функция `PhGetRects()` получает набор прямоугольников, и функция `PhGetData()` – порцию данных для события.

- ☞ Регион может накапливать данное событие, только если часть региона пересекает событие, и регион чувствителен к этому типу события.

Сжатие событий

Менеджер Photon'a сжимает события перетаскивания, границ и мыши. То есть, если висит событие этого типа, когда поступает другое событие, новое событие будет подсоединено к необработанным событиям. В результате приложение видит для этих событий только самые последние значения и избегает от накопления слишком большого количества ненужных событий.

Перетаскивание

Если Вам необходимо зафиксировать координаты мыши, например, для перетаскивания графических объектов в Вашем приложении, Вам понадобится работать с событиями.

- ☞ Если Вам надо перенести произвольные данные внутри Вашего приложения или между приложениями, почитайте главу "Тащи и бросай".

Имеется два типа перетаскивания:

- контурное перетаскивание Пользователь при перетаскивании видит контур. Когда перетаскивание завершено, приложение переставляет виджет.
- непрозрачное перетаскивание Приложение перемещает виджет как продвигающееся перетаскивание.

Перетаскивание выполняется в два этапа:

1. Инициализация перетаскивания, обычно когда пользователь щёлкает мышью на чём-то.
2. Обработка событий перетаскивания (`Ph_EV_DRAG`).

Эти шаги обсуждаются в нижеследующих подразделах.

Инициализация перетаскивания

Где Вы инициализируете перетаскивание, зависит от того, как пользователь собирается перетаскивать виджеты. Например, если пользователь удерживает нажатой левую кнопку мыши на виджете, чтобы его перетащить, перетаскивание инициализируется в ответной реакции `Arm (Pt_CB_ARM)` или `Outbound (Pt_CB_OUTBOUND)` виджета. Убедитесь, что в ресурсе `Pt_ARG_FLAGS` виджета установлен флаг `Pt_SELECTABLE`.

Перетаскивание начинается с вызова функции `PhInitDrag()`:

```
int PhInitDrag(PhRid_t      rid,
               unsigned     flags,
               PhRect_t     *rect,
               PhRect_t     *boundary,
               unsigned     input_group,
               PhDim_t      *min,
```

```

    PhDim_t          *max,
    const PhDim_t    *step,
    const PhPoint_t  *ptrpos,
    const PhCursorDescription_t *cursor );

```

где используемые аргументы:

<i>rid</i>	Идентификатор региона, с которым связаны <i>rect</i> и <i>boundary</i> . Вы можете получить его, вызвав функцию <code>PtWidgetRid()</code> .
<i>flags</i>	Указывает, будет ли использоваться контурное или непрозрачное перетаскивание, и какой край (края) перетаскиваемого прямоугольника оставляют след, что описано ниже.
<i>rect</i>	Структура <code>RhRect_t</code> (см. "Справочник библиотечных функций Photon'a"), которая определяет область перетаскивания.
<i>boundary</i>	Прямоугольная область, ограничивающая перетаскивание.
<i>input_group</i>	Получить это можно из события в параметре <i>cbinfo</i> ответной реакции, вызвав функцию <code>PhInputGroup()</code> .
<i>min, max</i>	Указатели на структуры типа <code>Ph_Dim_t</code> (см. "Справочник библиотечных функций Photon'a"), которые определяют минимальный и максимальный размеры перетаскиваемого прямоугольника.
<i>step</i>	Ступенчатость перетаскивания.
<i>ptrpos</i>	Если не <code>NULL</code> , то это указатель на структуру <code>PhPoint_t</code> (см. "Справочник библиотечных функций Photon'a"), который определяет начальную позицию курсора для перетаскивания. Приложения должны брать его от события, которое вызывает решение на начало перетаскивания. Если курсор переместился из этой позиции за время, когда Ваш <code>PhInitDrag()</code> достигнул Photon'a, Ваше перетаскивание соответствующим образом обновляется. Другими словами, Photon делает поведение перетаскивания таким, что оно как бы началось оттуда, где Вы думали был курсор, а не оттуда, где он был в действительности несколькими мгновениями позже.
<i>cursor</i>	Если не <code>NULL</code> , определяет, как курсор должен выглядеть при перетаскивании.

Если во *flags* включён `Ph_DRAG_TRACK`, используется непрозрачное перетаскивание; если `ph_DRAG_TRACK` не включён – контурное. Следующие флаги указывают, какой край (края) перетаскиваемого прямоугольника оставляет след:

- `Ph_TRACK_LEFT` – левый
- `Ph_TRACK_RIGHT` – правый
- `Ph_TRACK_TOP` – верхний
- `Ph_TRACK_BOTTOM` – нижний
- `Ph_TRACK_DRAG` – все вышеперечисленные

Контурное перетаскивание

Следующий пример демонстрирует ответную реакцию `Arm` (`Pt_CB_ARM`), которая инициализирует контурное перетаскивание:

```

/* Запуск перетаскивания виджета */

/* Стандартные хедеры */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

/* Инструментальные хедеры */
#include <Ph.h>
#include <Pt.h>
#include <Ap.h>

/* Локальные хедеры */
#include "globals.h"
#include "abimport.h"
#include "proto.h"

```

```

int start_dragging( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo ) {
    PhDim_t *dimension;
    PhRect_t rect;
    PhRect_t boundary;

    /* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    widget = widget, apinfo = apinfo, cbinfo = cbinfo;

    /* Установка перетаскиваемого прямоугольника в позицию и размеры перетаскиваемого виджета */
    PtWidgetExtent (widget, &rect);

    /* Установка границ перетаскивания по границам окна */

    PtGetResource (ABW_base, Pt_ARG_DIM, &dimension, 0);
    boundary.ul.x = 0;
    boundary.ul.y = 0;
    boundary.lr.x = dimension->w - 1;
    boundary.lr.y = dimension->h - 1;

    /* Инициализация контурного перетаскивания (Ph_DRAG_TRACK не задано) */

    PhInitDrag (PtWidgetRid (ABW_base),
                Ph_TRACK_DRAG,
                &rect, &boundary,
                PhInputGroup( cbinfo->event ),
                NULL, NULL, NULL, NULL, NULL );

    /* Сохранение указателя на перетаскиваемый виджет */
    dragged_widget = widget;

    return( Pt_CONTINUE );
}

```

Вышеописанная ответная реакция добавляется к ответной реакции Arm (Pt_CB_ARM) перетаскиваемого виджета. Это может быть использовано для перетаскивания любого виджета, так что указатель на виджет сохраняется в глобальной переменной *dragged_widget*.

Непрозрачное перетаскивание

Если Вы хотите использовать непрозрачное перетаскивание, добавьте к вызову PhInitDrag() флаг Ph_DRAG_TRACK():

```

PhInitDrag( PtWidgetRid (ABW_base),
            Ph_TRACK_DRAG | Ph_DRAG_TRACK,
            &rect, &boundary,
            PhInputGroup( cbinfo->event ),
            NULL, NULL, NULL, NULL, NULL );

```

Обработка событий перетаскивания

Чтобы обработать события перетаскивания (Ph_EV_DRAG), Вам надо определить необработанную (Pt_CB_RAW) или отфильтрованную (Pt_CB_FILTER) ответную реакцию.

- ☞ Необработанная или отфильтрованная ответная реакция должна быть определена для виджета, чей регион передаётся функции PhInitDrag(), а не для перетаскиваемого виджета. В данном примере необработанная ответная реакция определена для базового окна.

Как описано в разделе "Обработка событий – необработанные и отфильтрованные ответные реакции" в главе "Редактирование ресурсов и ответных реакций в PhAB", чтобы указать, для каких событий вызываются ответные реакции, Вы используете маску событий. Для перетаскивания событием является Ph_EV_DRAG. Наиболее часто используемыми подтипами для этого события являются:

Ph_EV_DRAG_START	Пользователь начал перетаскивание.
Ph_EV_DRAG_MOVE	Перетаскивание выполняется (только для непрозрачного перетаскивания).
Ph_EV_DRAG_COMPLETE	Пользователь отпустил кнопку мыши.

Контурное перетаскивание

Если Вы выполняете контурное перетаскивание, подтипом интересующего Вас события является `Ph_EV_DRAG_COMPLETE`. Когда случается событие, Ваша ответная реакция должна:

1. Получить данные, связанные с событием. Они представляют из себя структуру `PhDragEvent_t`, включающую в себе месторасположение перетаскиваемого прямоугольника, в абсолютных координатах. Для более полной информации см. "Справочник библиотечных функций Photon'a".
2. Вычислить новую позицию виджета относительно перетаскиваемого региона. Это позиция верхнего левого угла перетаскиваемого прямоугольника, пересчитанная на величину, заданную в области *translation* события.
3. Установить ресурс `Pt_ARG_POS` виджета в новую позицию.

☞ Помните, что параметр *widget* ответной реакции является указателем на контейнер (базовое окно в нашем примере), а не на перетаскиваемый виджет. Убедитесь, что передали правильный виджет функции `PtSetResources()` или `PtSetResource()`, когда устанавливали ресурс `Pt_ARG_POS`.

Например, вот необработанная ответная реакция для контурного перетаскивания, инициализированного выше:

```
/* необработанная ответная реакция для обработки событий перетаскивания;
   Задайте её для базового окна */

/* Стандартные хедеры */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

/* Инструментальные хедеры */
#include <Ph.h>
#include <Pt.h>
#include <Ap.h>

/* Локальные хедеры */
#include "globals.h"
#include "abimport.h"
#include "proto.h"

int end_dragging( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo ) {
    PhDragEvent_t *dragData;
    PhPoint_t new_pos;

    /* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    widget = widget, apinfo = apinfo, cbinfo = cbinfo;

    /* Игнорируем все события до тех пор, пока не выполнится перетаскивание */
    if (cbinfo->event->subtype != Ph_EV_DRAG_COMPLETE) {
        return (Pt_CONTINUE);
    }

    /* Получаем данные, связанные с событием */
    dragData = PhGetData (cbinfo->event);

    /* Прямоугольник в этих данных представляет из себя перетаскиваемый прямоугольник
       в абсолютных координатах. Мы хотим вычислить новую позицию виджета относительно
       перетаскиваемого региона */
    new_pos.x = dragData->rect.ul.x + cbinfo->event->translation.x;
    new_pos.y = dragData->rect.ul.y + cbinfo->event->translation.y;
    printf ("Новая позиция: (%d, %d)\n", new_pos.x, new_pos.y);

    /* Перемещаем виджет */
    PtSetResource (dragged_widget, Pt_ARG_POS, &new_pos, 0);

    return( Pt_CONTINUE );
}
```

Непрозрачное перетаскивание

Ответная реакция в случае непрозрачного перетаскивания сходна с подобной при контурном перетаскивании: единственное отличие – это подтип обработанного события:

```
if (cbinfo->event->subtype != Ph_EV_DRAG_MOVE) {  
    return (Pt_CONTINUE);  
}
```

Глава 24. Управление окнами

Иногда Вам требуется взаимодействовать с оконным менеджером Photon'a, чтобы делать поведение Ваших окон и диалогов таким, каким бы Вам хотелось.

В этой главе обсуждается:

- Флаги управления окнами
- Уведомительная ответная реакция
- Получение и установка состояния окна
- Управление несколькими окнами
- Функции управления окнами
- Исполнение самостоятельного приложения
- Модальные диалоги

☞ Помните, что оконные и диалоговые модули PhAB'a реализованы как виджеты типа PtWindow. Тип PtWindow имеет много ресурсов, используемых для осуществления взаимодействия с оконным менеджером.

Информацию по регионам оконного менеджера см. в приложении "Архитектура Photon'a". Список относящихся к этому функций см. в разделе "Оконный менеджер" главы "Сводка функций" "Справочника библиотечных функций Photon'a".

Флаги управления окнами

Виджет PtWindow определяет различные типы флагов:

Pt_ARG_WINDOW_RENDER_FLAGS	Какая отделка окна появляется на оконной рамке
Pt_ARG_WINDOW_MANAGED_FLAGS	Как оконный менеджер работает в окне
Pt_ARG_WINDOW_NOTIFY_FLAGS	О каких событиях оконного менеджера хотело бы получать уведомление Ваше приложение
Pt_ARG_WINDOW_STATE	Текущее состояние окна

☞ Если Вы изменили состояние окна после его реализации, Вы должны знать об этом оконном менеджеру. См. раздел "Получение и установка состояния окна" ниже в этой главе.

Флаги отображения окна

Ресурс Pt_ARG_WINDOW_RENDER_FLAGS задаёт, что появляется на рамке окна.

Чтобы отобразить:	Установите этот бит:	Умолчание
Рамку	Ph_WM_RENDER_BORDER	Да
Ручки изменения размеров	Ph_WM_RENDER_RESIZE	Да
Заголовочный брусок рамки	Ph_WM_RENDER_TITLE	Да
Кнопку меню	Ph_WM_RENDER_MENU	Да
Кнопку закрытия	Ph_WM_RENDER_CLOSE	
Кнопку помощи (значок вопроса)	Ph_WM_RENDER_HELP	
Кнопку минимизации	Ph_WM_RENDER_MIN	Да
Кнопку максимизации	Ph_WM_RENDER_MAX	Да
Кнопку сворачивания	Ph_WM_RENDER_COLLAPSE	Да
Дополнительную линию внутри стандартных границ	Ph_WM_RENDER_INLINE	

- ☞ Использование этих флагов для отображения элементов отделки не приводит к тому, что оконный менеджер делает что-то с этими элементами. Вам может понадобиться установить флаги управления окном и/или флаги уведомления.

Флаги управления окном

Ресурс `Pt_ARG_WINDOW_MANAGED_FLAGS` задаёт, какие действия должен обрабатывать оконный менеджер:

Чтобы позволить оконному менеджеру:	Установите этот бит:	Умолчание:
Закрывать окно	<code>Ph_WM_CLOSE</code>	Да
Дать фокус	<code>Ph_WM_FOCUS</code>	Да
Построить и управлять оконным меню	<code>Ph_WM_MENU</code>	Да
Переместить окно вперёд	<code>Ph_WM_TOFRONT</code>	Да
Переместить окно назад	<code>Ph_WM_TOBACK</code>	Да
Переместить окно на новую консоль, как будто пользователь переключил консоли	<code>Ph_WM_CONSWITCH</code>	
Изменить размеры окна	<code>Ph_WM_RESIZE</code>	Да
Переместить окно	<code>Ph_WM_MOVE</code>	Да
Скрыть (т.е. минимизировать) окно	<code>Ph_WM_HIDE</code>	Да
Максимизировать окно	<code>Ph_WM_MAX</code>	Да
Отобразить окно как фон	<code>Ph_WM_BACKDROP</code>	
Восстановить окно	<code>Ph_WM_RESTORE</code>	Да
Обеспечить контекстно-чувствительной помощью	<code>Ph_WM_HELP</code>	
Сделать окно принудительно передним	<code>Ph_WM_FFRONT</code>	
Свернуть окно в планку заголовка	<code>Ph_WM_COLLAPSE</code>	
Защитить Вас от заикливания фокуса в окне посредством <code>Alt-Esc</code> , <code>Alt-Shift-Esc</code> или <code>Alt-Tab</code>	<code>Ph_WM_NO_FOCUS_LIST</code>	

По умолчанию, выбранными являются флаги в соответствии с набором, определённым в `Ph_WM_APP_DEF_MANAGED` в `<PhWm.h>`. Вам надо выключить флаги управления, если Вы:

- не хотите, чтобы происходила соответствующая операция
- хотите, чтобы соответствующая операция обрабатывалась приложением. В этом случае Вам понадобится также установить соответствующий флаг уведомления.

Оконные флаги уведомления

Ресурс `Pt_ARG_WINDOW_NOTIFY_FLAGS` определяет, о какой операции управления окнами должно быть уведомлено Ваше приложение. Этот ресурс использует те же самые биты, что и `Pt_ARG_WINDOW_MANAGED_FLAGS`:

Быть уведомленным, когда:	Установлен этот бит:	Умолчание:
Окно было закрыто (см. ниже)	<code>Ph_WM_CLOSE</code>	Да
Окно получило/потеряло фокус	<code>Ph_WM_FOCUS</code>	
Оконное меню было запрошено или выключено	<code>Ph_WM_MENU</code>	
Окно было перемещено вперёд	<code>Ph_WM_TOFRONT</code>	
Окно было перемещено назад	<code>Ph_WM_TOBACK</code>	
Окно переключило консоли	<code>Ph_WM_CONSWITCH</code>	
Размеры окна были изменены	<code>Ph_WM_RESIZE</code>	Да
Окно было перемещено	<code>Ph_WM_MOVE</code>	
Окно было скрыто или обратно показано	<code>Ph_WM_HIDE</code>	
Окно было максимизировано	<code>Ph_WM_MAX</code>	

Окно было сделано фоновым	Ph_WM_BACKDROP	
Окно было восстановлено	Ph_WM_RESTORE	
Была нажата кнопка помощи	Ph_WM_HELP	Да
Окно было сделано принудительно передним или это было отключено	Ph_WM_FFRONT	

Принимаемым по умолчанию набором является

Ph_WM_RESIZE | Ph_WM_CLOSE | Ph_WM_HELP.

Когда происходит запрошенное действие, вызывается ответная реакция Pt_CB_WINDOW. См. раздел "Ответная реакция уведомления" ниже. Если Вы установили флаг уведомления Ph_WM_CLOSE, ответная реакция Pt_CB_WINDOW Вашего приложения вызывается, когда кто-то хочет закрыть окно. Ваше приложение не закрывает окно – оно может решить, что его надо оставить открытым. Напротив, ответная реакция Pt_CB_WINDOW_CLOSING вызывается, когда окно удалено (unrealized), но до того, как удалён его регион. На этот момент приложение не может прекратить закрытие окна.

☞ Если вы установили флаг управления Ph_WM_CLOSE, менеджер окна указывает на необходимость обработки закрытия окна. В этом случае вызывается ответная реакция Pt_CB_WINDOW_CLOSING, но не ответная реакция Pt_CB_WINDOW.

Ответная реакция уведомления

Когда происходит действие оконного менеджера, которое занесено в список уведомительных флагов окна (Pt_ARG_WINDOW_NOTIFY_FLAGS), вызывается ответная реакция окна Pt_CB_WINDOW.

Каждой функции ответной реакции, имеющейся в списке этого ресурса, передаётся структура PtCallbackInfo(см. "Справочник виджетов Photon'a"), которая содержит по меньшей мере следующие члены:

<i>reason</i>	Pt_CB_WINDOW
<i>reason_subtype</i>	0 (не используется)
<i>event</i>	Указатель на событие, вызвавшее ответную реакцию
<i>cbdata</i>	Указатель на структуру PhWindowEvent_t (описанную в "Справочнике библиотечных функций Photon'a")

Эти функции ответных реакций должны возвращать Pt_CONTINUE.

Пример: проверка закрытия окна

Предположим, Вы хотите проверить, действительно ли пользователь хочет завершить работу приложения, закрывая окно. Вот что Вам надо сделать:

- Сбросить флаг Ph_WM_CLOSE в ресурсе Pt_ARG_WINDOW_MANAGED_FLAGS. Это укажет оконному менеджеру не закрывать окно.
- Установите флаг Ph_WM_CLOSE в ресурсе Pt_ARG_WINDOW_NOTIFY_FLAGS. Оконный менеджер будет уведомлять Вас, когда пользователь попытается закрыть окно.
- Добавьте ответную реакцию Pt_CB_WINDOW такого вида:

```
int window_callback( PtWidget_t *widget, ApInfo_t *apinfo, PtCallbackInfo_t *cbinfo ) {
    PhWindowEvent_t *we = cbinfo->cbdata;
    char *btns[] = { "&Yes", "&No" };
    char Helvetica14[MAX_FONT_TAG];

    /* предотвращает предупреждения (варнинги) об отсутствии ссылок */
    widget = widget, apinfo = apinfo, cbinfo = cbinfo;
```

```

if ( we->event_f == Ph_WM_CLOSE ) {

    /* Спросит пользователя, действительно ли он хочет выйти.
       Использовать шрифт 14-point Helvetica, если он доступен */

    switch( PtAlert( ABW_base, NULL, NULL, NULL, "Вы действительно хотите уйти?",
                   PfGenerateFontName("Helvetica", 0, 14, Helvetica14),
                   2, btns, NULL, 1, 2, Pt_MODAL ) ) {

        case 1: /* да */
            PtExit (EXIT_SUCCESS);
            break;
        case 2: /* нет */
            return (Pt_CONTINUE);
    }
}
else {
    /* Проверка других событий */
}

return( Pt_CONTINUE );
}

```

Есть весьма существенная разница между событием Ph_WM_CLOSE и ответной реакцией закрытия окна (Pt_CB_WINDOW_CLOSING).

Ответная реакция Pt_CB_WINDOW с событием Ph_WM_CLOSE является просто уведомлением со стороны оконного менеджера Photon'a, что пользователь щёлкнул на кнопке закрытия или выбрал пункт "Close" из меню оконного менеджера Photon'a. Если в ресурсе Pt_ARG_WINDOW_MANAGED_FLAGS сброшен бит Ph_WM_CLOSE, библиотека больше не предпринимает никаких действий.

Закрытие окна (Pt_CB_WINDOW_CLOSING) вызывается, когда окно собирается по какой-либо причине прекратить своё существование (unrealize). Это включает транспортировку в другой Photon и явные вызовы PtDestroyWidget() или PtUnrealizeWidget(). Если в ответной реакции закрытия окна Вы хотите убедиться, что окно действительно уничтожено, установите флаг Pt_DESTROYED в Pt_ARG_FLAGS. Вы можете также использовать ответную реакцию Pt_CB_DESTROYED, чтобы знать, когда окно помечается для уничтожения, или Pt_CB_IS_DESTROYED, чтобы знать, когда оно уничтожается.

Заметьте также, что вызов функции exit() явным образом игнорирует все эти ответные реакции.

Получение и установка состояния окна

Ресурс Pt_ARG_WINDOW_STATE управляет состоянием окна:

Чтобы сделать это:	Установите этот бит:
Максимизировать окно	Ph_WM_STATE_ISMAX
Сделать окно фоновым	Ph_WM_STATE_ISBACKDROP
Минимизировать окно	Ph_WM_STATE_ISHIDDEN
Разместить базовое окно впереди окон всех других приложений	Ph_WM_STATE_ISFRONT
Передать фокус клавиатуры на окно, если фокус курсора отключён	Ph_WM_STATE_ISFOCUS
Передать приложению комбинации с клавишей <Alt>	Ph_WM_STATE_ISALTKEY
Заблокировать окно	Ph_WM_STATE_ISBLOCED (только для чтения)

Принятым по умолчанию значением является PH_WM_STATE_ISFOCUS.

☞ Вы можете получить и установить состояние окна в любой момент, используя ресурс Pt_ARG_WINDOW_STATE, но это может привести к непредсказуемым результатам, если пользователь как раз в этот момент изменит состояние окна.

Наиболее безопасным временем, когда можно использовать этот ресурс для установки состояния окна – время перед тем, как окно реализовано. Например, Вы можете выполнить установки, когда создаётся виджет PtWindow или в ответной реакции окна Pt_CB_WINDOW_OPENING. Установки обретут силу, когда окно реализуется.

Вы можете установить Pt_ARG_WINDOW_STATE после реализации окна, основывая Ваши изменения на Ваших предположениях о том, каково текущее состояние окна, но безопаснее сообщить оконному менеджеру о том, как Вы хотите изменить состояние окна, вызвав:

PtForwardWindowEvent()	Изменить состояние окна, связанного с данным идентификатором региона
PtForwardWindowTaskEvent()	Изменить состояние окна, связанного с данным идентификатором коннекции Photon'a.

Например, чтобы минимизировать окно, которое принадлежит Вашему приложению и уже открыто:

```
static void send_key( long key ) {
    struct {
        PhEvent_t event;
        PhRect_t rect;
        PhKeyEvent_t pevent;
    } new_event;

    PhEvent_t event;
    PhKeyEvent_t key_event;

    PhRect_t rect;

    rect.ul.x = rect.ul.y = 0;
    rect.lr.x = rect.lr.y = 0;

    memset( &event, 0, sizeof(event));
    memset( &key_event, 0, sizeof(key_event) );

    event.type = Ph_EV_KEY;
    event.emitter.rid = Ph_DEV_RID;
    event.num_rects = 1;
    event.data_len = sizeof(key_event);
    event.input_group = 1;

    key_event.key_cap = key;
    key_event.key_sym = key;

    if ( isascii( key ) && isupper( key ) ) {
        key_event.key_mods = Pk_KM_Shift;
    }

    /* Генерирование нажатия клавиши */

    key_event.key_flags = Pk_KF_Sym_Valid | Pk_KF_Cap_Valid | Pk_KF_Key_Down;
    PhEmit( &event, &rect, &key_event );

    /* Генерирование отпускания клавиши */

    key_event.key_flags &= ~(Pk_KF_Key_Down | Pk_KF_Sym_Valid);
    PhEmit( &event, &rect, &key_event );

    return;
}
```

Чтобы минимизировать окно, принадлежащее другому приложению и уже открыто:

```
PhWindowEvent_t event;

memset( &event, 0, sizeof( event ) );
```

```
event.event_f = Ph_WM_HIDE;
event.event_state = Ph_WM_EVSTATE_HIDE;
PtForwardWindowTaskEvent( connection_id, &event );
```

- ☞ Когда Вы вызываете эти функции, Вы просите оконный менеджер выполнить заданное действие. Если это действие не входит в набор флагов управления (Pt_ARG_WINDOW_MANAGED_FLAGS) для данного окна, оконный менеджер этого и не сделает.

Управление несколькими окнами

Если Ваше приложение имеет более одного окна, Вам понадобится учитывать взаимодействие между ними.

По определению, окно-потомок *всегда* располагается перед своим родителем. Окна-потомки могут размещаться впереди и позади своих братьев. Для окон, способных переходить за другие окна, они должны быть братьями. Таким образом, окно, способное перемещаться за базовое окно, не должно иметь родителя.

Функции управления окнами

Нижеприведенные низкоуровневые функции связаны с оконным менеджером, но Вам не надо использовать их в приложении, использующем виджеты:

PhWindowChange()	Модифицирует атрибуты региона окна
PhWindowClose()	Закрывает окно
PhWindowOpen()	Создаёт регион окна

Эти функции могут быть вызваны в приложении, использующем виджеты:

PhWindowQueryVisible()	Запрашивает видимое пространство
PtConsoleSwitch()	Переключает на другую виртуальную консоль
PtForwardWindowEvent()	Отсылает событие окна
PtForwardWindowTaskEvent()	Отсылает событие окна задаче
PtWindowConsoleSwitch()	Переключает на консоль, на которой отображено заданное окно
PtWindowFrameSize()	Определяет размер рамки окна

Запуск самостоятельного приложения

Если предполагается, что приложение должно исполняться само, Вам может понадобиться:

- Сделать так, чтобы Ваше приложение заполнило экран. Установите Ph_WM_STATE_ISMAX в ресурсе Pt_ARG_WINDOW_STATE базового окна.
- Отключите все флаги во флагах Pt_ARG_WINDOW_RENDER_FLAGS базового окна, так чтобы у окна не было заголовочного бруска рамки, границ, кнопок меню и всего такого прочего – этого всего не нужно, если никакие другие приложения не будут исполняться.

[Примечание от переводчика: к сожалению, при этом базовое окно занимает всё доступное пространство, а не весь экран. Иными словами, всю область экрана, кроме Shelf'ов и Task Bar'a. Так что для того, чтобы перекрыть весь экран, можно установить в ресурсах окна Minimum Window Width = Maximum Window Width = 1015 (например), и аналогично по высоте (т.е. для разрешения 1024x768 можно 1015x740), и снять, как и указывается, по крайней мере, следующие флаги Ph_WM_RENDER_ : COLLAPSE, MIN, RESIZE, MAX, MENU, CLOSE].*

Модальные диалоги

Иногда Вам понадобится, чтобы перед продолжением своей работы программа запросила у пользователя какую-то информацию. Обычно это делается с помощью всплывающего диалога; если Вы не хотите, чтобы пользователь мог выбрать какое-либо другое действие перед тем, как предоставить эту информацию, Вы должны использовать *модальный диалог*.

Модальный диалог не позволяет вводу пользователя пройти на какой-либо иной виджет приложения. Используя для запроса информации модальный диалог, Вы обеспечиваете, что события обрабатываются внутри функции ответной реакции. Чтобы создать модальный диалог, Вы должны создать новый виджет типа `PtWindow`, обычно как потомка главного окна приложения.

Для активации модального диалога Вы должны реализовать диалоговый виджет и заблокировать все остальные оконные виджеты приложения. Чтобы заблокировать окно или окна, вызовите одну из функций:

<code>PtBlockAllWindows()</code>	Блокировать все окна, кроме одного с заданным виджетом
<code>PtBlockWindow()</code>	Блокировать заданное окно

Обе эти функции возвращают список заблокированных виджетов, который Вам понадобится, когда Вы будете их разблокировать. В отличие от блокировки окон, Вы можете сделать диалог модальным с помощью функции `PtMakeModal()`.

После того как модальный диалог активирован, вызовите функцию `PtModalBlock()`, чтобы запустить модальную петлю обработки событий Photon'a вплоть до возникновения состояния завершения.

Когда операция, связанная с модальным диалогом, завершена или прервана, Вам надо убрать диалог. Чтобы это сделать:

1. Вызовите функцию `PtModalUnblock()`, чтобы остановить выполнение модальной петли. Вы можете установить значение, возвращённое функцией `PtModalBlock()`.
2. Уничтожьте (`destroy`) или удалите (`unrealize`) сам диалог.

Вызовите функцию `PtUnblockWindows()`, чтобы разблокировать все оконные виджеты, которые были заблокированы при создании диалога. Вам не надо это делать, если вместо функций `PtBlockAllWindows()` или `PtBlockWindow()` Вы вызывали функцию `PtMakeModal()`.

Мы можем легко изменить наш предыдущий пример рабочей процедуры, так что её "прогрессный" диалог будет вести себя как модальный диалог. Мы добавим структуру `PtModalCtrl_t` в ответную реакцию закрытия окна для её использования функциями `PtModalBlock()` и `PtModalUnblock()`.

Ответная реакция `done()` вместо освобождения закрытия окна вызывает функцию `PtModalUnblock()`:

```
int done(PtWidget_t *w, void *client, PtCallbackInfo_t *call) {
    CountdownClosure *closure = (CountdownClosure *)client;

    call = call;

    if (!closure->done) {
        PtAppRemoveWorkProc(NULL, closure->work_id);
    }
    PtDestroyWidget(closure->dialog->widget);
    free(closure->dialog);

    /* Новое: завершаем модальную петлю, возвращаем значение счётчика в качестве ответа */
    PtModalUnblock(&(closure->modal_control), (void *) &(closure->value));
    return (Pt_CONTINUE);
}
```

Всё, что осталось в этом месте сделать – это изменить функцию ответной реакции `push_button_cb()`, так чтобы она блокировала окно после реализации "прогрессного" диалога, запускала модальную петлю, разблокировала окна и освобождала закрытие после исчезновения диалога.

Вот новая версия функции ответной реакции `push_button_cb()`:

```
int push_button_cb(PtWidget_t *w, void *client, PtCallbackInfo_t *call) {
PtWidget_t *parent = (PtWidget_t *)client;
WorkDialog *dialog;
PtBlockedList_t * blocked_list;
void * response;

w = w; call = call;

dialog = create_working_dialog(parent);

if (dialog) {
    CountdownClosure *closure = (CountdownClosure *) malloc(sizeof(CountdownClosure));

    if (closure) {
        PtWorkProcId_t *id;

        closure->dialog = dialog;
        closure->value = 0;
        closure->maxvalue = 200000;
        closure->done = 0;
        closure->work_id = id =
            PtAppAddWorkProc(NULL, count_cb, closure);

        PtAddCallback(dialog->ok_button, Pt_CB_ACTIVATE, done, closure);
        PtRealizeWidget(dialog->widget);

        /* Новое: блокируем все окна, кроме диалога, обрабатываем события вплоть
           до закрытия диалога, и затем разблокируем все окна */

        locked_list = PtBlockAllWindows (dialog->widget,
                                         Ph_CURSOR_NOINPUT, Pg_TRANSPARENT);

        response = PtModalBlock( &(closure->modal_control), 0 );
        printf ("Достигнутое значение равно %d\n", *(int *)response );
        free (closure);

        PtUnblockWindows (blocked_list);

    }
}
return (Pt_CONTINUE);
}
```

А вот новая версия программы в целом:

```
#include <Pt.h>

typedef struct workDialog {
    PtWidget_t *widget;
    PtWidget_t *label;
    PtWidget_t *ok_button;
} WorkDialog;

typedef struct countdownClosure {
    WorkDialog *dialog;
    int value;
    int maxvalue;
    int done;
    PtWorkProcId_t *work_id;

    /* Новый член: */
    PtModalCtrl_t modal_control;
} CountdownClosure;

WorkDialog *create_working_dialog(PtWidget_t *parent) {
PhDim_t dim;
PtArg_t args[3];
int nargs;
PtWidget_t *window, *group;
```

```

WorkDialog *dialog = (WorkDialog *)malloc(sizeof(WorkDialog));

if (dialog) {
    nargs = 0;
    PtSetArg(&nargs[nargs], Pt_ARG_WIN_PARENT, parent, 0);
    nargs++;
    PtSetParentWidget(NULL);
    dialog->widget = window = PtCreateWidget(PtWindow, parent, nargs, args);

    nargs = 0;
    PtSetArg(&nargs[nargs], Pt_ARG_GROUP_ORIENTATION, Pt_GROUP_VERTICAL, 0);
    nargs++;
    PtSetArg(&nargs[nargs], Pt_ARG_GROUP_VERT_ALIGN, Pt_GROUP_VERT_CENTER, 0);
    nargs++;
    group = PtCreateWidget(PtGroup, window, nargs, args);

    nargs = 0;
    dim.w = 200;
    dim.h = 100;
    PtSetArg(&nargs[nargs], Pt_ARG_DIM, &dim, 0);
    nargs++;
    PtSetArg(&nargs[nargs], Pt_ARG_TEXT_STRING, "Counter:          ", 0);
    nargs++;
    dialog->label = PtCreateWidget(PtLabel, group, nargs, args);

    PtCreateWidget(PtSeparator, group, 0, NULL);

    nargs = 0;
    PtSetArg(&nargs[nargs], Pt_ARG_TEXT_STRING, "Stop", 0);
    nargs++;
    dialog->ok_button = PtCreateWidget(PtButton, group, 1, args);
}
return dialog;
} // create_working_dialog()

int done(PtWidget_t *w, void *client, PtCallbackInfo_t *call) {
    CountdownClosure *closure = (CountdownClosure *)client;

    call = call;

    if (!closure->done) {
        PtAppRemoveWorkProc(NULL, closure->work_id);
    }
    PtDestroyWidget(closure->dialog->widget);
    free(closure->dialog);

    /* Новое: завершаем модальную петлю, возвращаем значение счётчика в качестве ответа */
    PtModalUnblock(&(closure->modal_control), (void *) &(closure->value));

    return (Pt_CONTINUE);
} // done()

int count_cb(void *data) {
    CountdownClosure *closure = (CountdownClosure *)data;
    char buf[64];
    int finished = 0;

    if ( closure->value++ == 0 || closure->value % 1000 == 0 ) {
        sprintf(buf, "Счётчик: %d", closure->value);
        PtSetResource( closure->dialog->label, Pt_ARG_TEXT_STRING, buf, 0);
    }

    if ( closure->value == closure->maxvalue ) {
        closure->done = finished = 1;
        PtSetResource( closure->dialog->ok_button, Pt_ARG_TEXT_STRING, "Done", 0);
    }

    return finished ? Pt_END : Pt_CONTINUE;
} // count_cb()

int push_button_cb(PtWidget_t *w, void *client, PtCallbackInfo_t *call) {
    PtWidget_t *parent = (PtWidget_t *)client;
    WorkDialog *dialog;
    PtBlockedList_t * blocked_list;
    void * response;

    w = w; call = call;

    dialog = create_working_dialog(parent);

```

```

if (dialog) {
    CountdownClosure *closure = (CountdownClosure *) malloc(sizeof(CountdownClosure));

    if (closure) {
        PtWorkProcId_t *id;

        closure->dialog = dialog;
        closure->value = 0;
        closure->maxvalue = 200000;
        closure->done = 0;
        closure->work_id = id = PtAppAddWorkProc(NULL, count_cb, closure);

        PtAddCallback(dialog->ok_button, Pt_CB_ACTIVATE, done, closure);
        PtRealizeWidget(dialog->widget);

        /* Новое: блокируем все окна, кроме диалога, обрабатываем события вплоть до его закрытия,
           и затем разблокируем все окна */

        blocked_list = PtBlockAllWindows (dialog->widget,
                                           Ph_CURSOR_NOINPUT, Pg_TRANSPARENT);

        response = PtModalBlock( &(closure->modal_control), 0 );
        printf ("Достигнутое значение равно %d\n", *(int *)response );
        free (closure);

        PtUnblockWindows (blocked_list);

    }
}
return (Pt_CONTINUE);
} // push_button_cb()

int main(int argc, char *argv[]) {
    PhDim_t dim;
    PtArg_t args[3];
    int n;
    PtWidget_t *window;
    PtCallback_t callbacks[] = {{push_button_cb, NULL}};
    char Helvetica14b[MAX_FONT_TAG];

    if (PtInit(NULL) == -1) PtExit(EXIT_FAILURE);

    dim.w = 200;
    dim.h = 100;
    PtSetArg(&args[0], Pt_ARG_DIM, &dim, 0);
    if ((window = PtCreateWidget(PtWindow, Pt_NO_PARENT, 1, args)) == NULL) PtExit(EXIT_FAILURE);

    callbacks[0].data = window;
    n = 0;
    PtSetArg(&args[n++], Pt_ARG_TEXT_STRING, "Count Down...", 0);

    /* Используется 14-пунктовый жирный шрифт Helvetica, если он доступен */
    if (PfGenerateFontName("Helvetica", PF_STYLE_BOLD, 14, Helvetica14b) == NULL) {
        perror("Невозможно сгенерировать имя шрифта");
    }
    else {
        PtSetArg(&args[n++], Pt_ARG_TEXT_FONT, Helvetica14b, 0);
    }
    PtSetArg(&args[n++], Pt_CB_ACTIVATE, callbacks, sizeof(callbacks)/sizeof(PtCallback_t));
    PtCreateWidget(PtButton, window, n, args);

    PtRealizeWidget(window);

    PtMainLoop();
    return (EXIT_SUCCESS);
} // main()

```

Если Ваш диалог является автономным и Вам просто надо подождать его завершения, Вам может пригодиться такая функция:

ArModalWait() Обработать события Photon'a, пока заданный виджет не будет уничтожен.

Глава 25. Программирование в Photon'e без PhAB'a

Мы настоятельно рекомендуем при разработке приложений Photon'a пользоваться PhAB'ом – а эта глава для тех чудачков, которые упорно не используют PhAB.

В этой главе обсуждается следующее:

- Основные шаги
- Компилирование и линковка не PhAB'овского приложения
- Образец приложения
- Увязывание кода приложения с виджетами
- Полный пример приложения

Основные шаги

Все приложения, использующие библиотеку виджетов Photon'a, выполняют одну и ту же базовую последовательность действий:

1. Включить файл <Pt.h> – стандартный заголовочный файл для библиотеки виджета.
2. Инициализировать инструментальные средства виджетов Photon'a, вызвав функцию PtInit() (или PtAppInit()), которая также создаёт основное окно).
3. Создать виджеты, поддерживающие интерфейс с пользователем, вызвав функцию PtCreateWidget. Эта функция может создавать новые виджеты в потомках заданного виджета или текущего контейнера, или виджеты, не имеющие родителя.
4. Зарегистрировать какие-либо ответные реакции в приложении с соответствующими виджетами, используя функции PtAddCallback() или PtAddCallbacks().
5. Реализовать виджеты, вызвав функцию PtRealizeWidget(). Эту функцию необходимо вызывать в приложении только один раз. Шаг реализации в действительности создаёт некие регионы Photon'a, которые затем назначаются и отображаются на экран. Пока этот шаг не отработан, никаких регионов не существует, и на экране ничего не отображается.
6. Обработать события Photon'a, вызвав функцию PtMainLoop(). На этом шаге инструментальные средства виджетов Photon'a берут на себя управление приложением и виджетами. Если какие-либо виджеты вызывают функции Вашего приложения, они должны быть предварительно зарегистрированы как ответные реакции.

Компилирование и линковка не PhAB'овского приложения

Чтобы скомпилировать и запустить на исполнение приложение, которое использует библиотеку виджетов Photon'a, Вы должны подлинковаться к главной библиотеке Photon'a ph и к библиотеке отображения phrender. Существует статическая и совместно используемая версии этих библиотек.



Библиотека photon предназначена только для приложений, созданных в версии 1.14 микроGUI Photon'a. Не комбинируйте эту библиотеку с текущими библиотеками или заголовочными файлами, в противном случае Ваше приложение будет исполняться неверно.

Мы рекомендуем, чтобы Вы всегда подлинковывались к библиотеке *совместного доступа*. Это позволит Вам иметь приложение меньшим по размеру и позволит ему наследовать новые

возможности, добавляемые к библиотеке виджетов при инсталляции новых реализаций библиотеки совместного доступа.

Библиотека Photon'a включает часть функций и определений виджетов. Если Ваше приложение использует функции Al (переводные) или Px (расширенные), Вам также понадобится подлинковаться к библиотеке rhexlib. Если ваше приложение использует функции Ap (PhAB'овские), Вам также надо подлинковаться к библиотеке Ap. Имена статических и совместно используемых библиотек одни и те же. По умолчанию qcc линкуется с библиотеками совместного доступа; чтобы линковаться со статическими библиотеками, задайте для qcc опцию `-Bstatic`. Например, если у Вас есть приложение по имени `hello.c`, командой на компилирование и линковку с библиотекой совместного доступа является:

```
qcc -o hello hello.c -lph -lphrender
```

Чтобы подлинковать статические библиотеки, команда должна быть такой:

```
qcc -o hello hello.c -Bstatic -lph -lphrender
```

☞ Совместно используемая библиотека `ph` не включает ничего, что требует операций с плавающей запятой (в текущей версии именно виджет `PtNumericFloat`), в то время как статическая библиотека включает. Чтобы слинковать приложение, включающее виджет `PtNumericFloat`, Вы можете линковать его только со статической библиотекой или сделать так:

```
qcc -o hello hello.c lph -Bstatic -lph -Bdynamic -lphrender
```

Образец приложения

Следующий пример демонстрирует простейшее приложение, использующее библиотеку виджетов. Программа создаёт окно, содержащее одну кнопку.

```
/* File: hello.c */
#include <Pt.h>

int main( int argc, char *argv[] ) {
    PtWidget_t *window;
    PtArg_t      args[1];

    if (PtInit(NULL) == -1) PtExit(EXIT_FAILURE);

    window = PtCreateWidget(PtWindow, Pt_NO_PARENT, 0, NULL);

    PtSetArg(&args[0], Pt_ARG_TEXT_STRING, "Нажмите для выхода", 0);
    PtCreateWidget(PtButton, window, 1, args);
    PtRealizeWidget(window);

    PtMainLoop();
    return (EXIT_SUCCESS);
}
```

Что происходит

Хотя это и простое приложение, в каждом из этих вызовов выполняется масса работы.

PtInit()

Функция `PtInit()` вызывает функцию `PhAttach()`, чтобы прикрепить к серверу Photon'a канал, и затем вызывает библиотеки виджета.

PtCreateWidget() – первый вызов

Первый вызов функции PtCreateWidget() создаёт оконный виджет, который взаимодействует с оконным менеджером и служит родителем остальным виджетам, созданным в приложении. Аргументами этой функции являются:

- класс создаваемого виджета (в нашем случае PtWindow)
- родитель виджета (Pt_NO_PARENT, поскольку окно не имеет родителя)
- количество элементов в списке аргументов
- список аргументов начальных значений для ресурсов виджета

Функция PtCreateWidget() возвращает указатель на созданный виджет. Более подробную информацию см. в разделе "Создание виджетов" в главе "Управление виджетами из программного кода приложения". Более полную информацию о виджетах и их ресурсах см. в "Справочнике виджетов Photon'a".

PtSetArg()

Макрос PtSetArg() устанавливает список аргументов, используемый для инициализации ресурсов кнопки при её создании. Более подробно см. в главе "Управление ресурсами в программном коде приложения".

PtCreateWidget() – второй вызов

Все виджеты приложения – кроме окна верхнего уровня – имеют контейнерный виджет в качестве родителя. Контейнерные виджеты могут содержать в себе другие контейнеры. Создание виджетов в приложении приводит к возникновению иерархии, называемой *семейством виджетов*.

Второй вызов функции PtCreateWidget() создаёт виджет кнопки как потомка оконного виджета, используя для инициализации ресурсов кнопки список аргументов. Вы можете передать в качестве родителя Pt_DEFAULT_PARENT, чтобы сделать виджет потомком контейнерного виджета, созданного самым последним; в этом случае результат будет тем же самым.

PtRealizeWidget()

Функция PtRealizeWidget() отображает виджет и всех его потомков в семействе виджетов. Наше простое приложение вызывает PtRealizeWidget() для окна верхнего уровня, так что отображаются все виджеты приложения.

Когда виджет реализован, он использует значения своих ресурсов, чтобы определить, насколько большим он должен быть, чтобы отобразить своё содержание. Перед реализацией виджета Вы должны установить все ресурсы, которые могут сказаться на размере. Вы можете изменить некоторые из этих ресурсов уже после того, как виджет реализован, но это будет только тогда, когда виджет определит, что он может или хочет изменить свои размеры, чтобы подстроиться под изменения в значении ресурса.

Вы можете установить флаги изменения размеров, которые виджет использует для того, чтобы определить, подстраивать ли ему или не подстраивать свои размеры в ответ на эти изменения, но заметьте, что если виджет выйдет за пределы размеров, выделенных ему его родителем, он будет отсечён в соответствии с размерами родителя. Не существует механизма, позволяющего виджету договориться со своим родителем о выделении большего пространства. Более подробно об этом см. в главе "Управление геометрией".

Если для корректного отображения требуется *регион* Photon'a, он создаётся каждый раз при реализации виджета. Регион требуется при *любом* из следующих условий:

- Виджет устанавливает курсор

- Виджету требуется получать события, который не перенаправляются ему его родительским контейнером (напр., охват границей, события перемещения указателя).
- В ресурсе `Pt_ARG_FLAGS` установлен флаг `Pt_REGION` (см. описание `PtWidget` в "Справочнике виджетов Photon'a").

Вы можете дереализовать (`unrealize`) виджет, вызвав функцию `PtUnrealizeWidget()`. Это влияет на факт отображения виджета и его потомков на экране, но не освобождает иерархию семейства виджетов. Вы позже вновь можете отобразить виджет, вызвав функцию `PtRealizeWidget()`.

Вы можете предотвратить реализацию виджета и его потомков при реализации его родителя. Чтобы это сделать, установите флаг `Pt_DELAY_REALIZE` в ресурсе `Pt_ARG_FLAGS`. Если Вы установите этот флаг, Вашей заботой станет вызвать функцию `PtRealizeWidget()` для этого виджета, когда Вы захотите, чтобы он появился на экране.

PtMainLoop()

Вызов функции `PtMainLoop` передаёт управление приложениям библиотеки виджетов Photon'a.

Библиотека виджетов ожидает события Photon'a и передаёт их виджетам для обработки. Программный код приложения исполняется только тогда, когда в результате появления события вызываются функции ответных реакций, которые приложение зарегистрировало в виджете для этого события.

Подсоединение программного кода приложения к виджету

Если вы откомпилируете, слинкуете и запустите образец приложения, Вы увидите, что появится окно с кнопкой на нём. Если Вы нажмёте на кнопку, ничего не произойдёт, поскольку с этим не связан никакой программный код приложения.

Библиотека виджетов Photon'a спроектирована таким образом, что код пользовательского интерфейса может содержаться совершенно отдельно от программного кода приложения. Пользовательский интерфейс скомпонован из кода, направленного на создание и управление иерархией семейства виджетов, и должен вызывать программный код приложения, чтобы отзываться на конкретные события или действия пользователя. Связь между программным кодом и пользовательским интерфейсом, позволяющая пользовательскому интерфейсу использовать программный код, является единственным местом, где эти две части получают глубокое знание друг о друге.

Связи между пользовательским интерфейсом и программным кодом приложения осуществляется использованием *ответных реакций* и *обработчиков событий*.

Ответные реакции – это особый тип ресурса виджета, который позволяет приложению реализовать все существующие возможности виджета. Используя ответные реакции, приложение может зарегистрировать функцию, которая будет затем вызываться библиотекой виджетов в ответ на конкретные ситуации, возникающие внутри виджета.

Обработчики событий (необработанные и отфильтрованные ответные реакции) обычно используются для добавления возможностей виджета. Например, Вы можете добавить поведение нажатия кнопки внутри виджета, который не имеет ответных реакций, связанных с событиями нажатия кнопки.

Ответные реакции

Ресурс ответной реакции используется для уведомления приложения о том, что у виджета произошло какое-то определённое действие (напр., Вы нажали кнопку). Каждый ресурс ответной реакции представляет некое действие пользователя, которое, по Вашей задумке, может быть интересно приложению.

Как и для всех ресурсов, виджет имеет свои ресурсы ответных реакций, определённые для его виджетного класса, и он наследует ресурсы ответных реакций, заданные для всех потомков его класса. Это означает, что виджет может иметь несколько действий пользователя, о которых он может уведомлять приложение.

Значением ресурса ответной реакции является список ответных реакций. Каждый элемент списка представляет из себя какую-то функцию приложения, вызываемую в ответ на изменение поведения и *данные клиента*, присоединённые к ответной реакции. Данные клиента – это указатель на какие-либо произвольного характера данные, которые могут понадобиться Вашему приложению, чтобы обеспечить правильную работу функции ответной реакции.

Информацию об ответных реакциях см. в разделе "Ответные реакции" главы "Управление виджетами в программном коде приложения".

Обработка событий

Когда мы создаём виджетный класс, мы не в силах предвидеть всего, что может понадобиться нашему приложению. Ваше приложение может захотеть быть уведомленным о чём-то, произошедшем с виджетом, что не связано с ресурсом ответной реакции. В этих случаях приложение может задать функции обработки событий.

Информацию об обработчиках событий см. в разделе "Обработчики событий" в главе "Управление виджетами в программном коде приложения".

Полный пример приложения

И теперь мы можем использовать наши вновь приобретённые знания о ресурсах и ответных реакциях для создания более функциональной версии образца приложения, данного выше. Используя ресурсы, мы можем дать виджету кнопки те же размеры, что и окну, и задать, какой шрифт использовать для текста надписи. Мы можем также задать ответную реакцию, исполняемую при нажатии кнопки. Мы сделаем ответную реакцию, которая будет отображать простое сообщение и завершаться.

Вот полный текст программного кода нашего образца программы с этими изменениями:

```
#include <stdio.h>
#include <stdlib.h>
#include <Pt.h>

int main( int argc, char *argv[] ) {
PtArg_t      args[3];
int          n;
PtWidget_t  *window;
int          push_button_cb( PtWidget_t *, void *, PtCallbackInfo_t * );
PtCallback_t callbacks[] = {{push_button_cb, NULL}};
char        Helvetic14[MAX_FONT_TAG];
```

```
if (PtInit(NULL) == -1) PtExit(EXIT_FAILURE);

window = PtCreateWidget(PtWindow, Pt_NO_PARENT, 0, NULL);

n = 0;
PtSetArg(&args[n++], Pt_ARG_TEXT_STRING, "Нажмите для выхода", 0);

/* Использовать, если доступен, шрифт 14-пунктовый жирный Helvetica */

if (PfGenerateFontName("Helvetica", 0, 14, Helvetica14) == NULL) {
    perror("Невозможно сгенерировать имя шрифта");
}
else {
    PtSetArg(&args[n++], Pt_ARG_TEXT_FONT, Helvetica14, 0);
}
PtSetArg(&args[n++], Pt_CB_ACTIVATE, callbacks, sizeof(callbacks)/sizeof(callbacks[0]));
PtCreateWidget(PtButton, window, n, args);

PtRealizeWidget(window);
PtMainLoop();
return (EXIT_SUCCESS);
}          // main()

int push_button_cb(PtWidget_t *w, void *data, PtCallbackInfo_t *cbinfo) {
    printf( "Я была нажата\n" );
    PtExit( EXIT_SUCCESS );
}

/* Эта строка никогда не будет исполняться, но она делает счастливым компилятор */

return( Pt_CONTINUE );
}
```

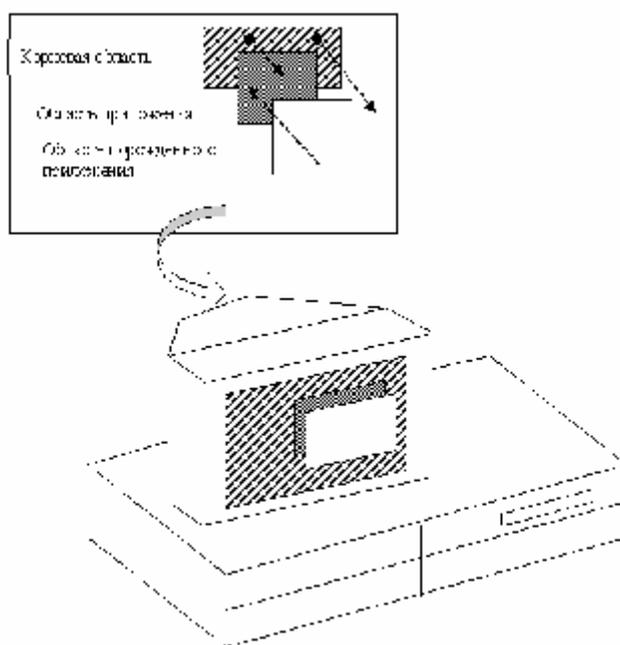
Приложение 1. Архитектура Photon'a

В этом приложении представлен технический обзор архитектуры Photon'a. Он включает:

- Пространство событий
- События
- Регионы
- Типы событий
- Как владельцы региона уведомляются о событиях
- Регион устройств
- Регион драйверов
- Оконный менеджер Photon'a

Пространство событий

Важнейшей характерной чертой Photon'a является способ представления графических приложений. Все приложения Photon'a состоят из одного или более прямоугольников, называемых *регионами*. Эти регионы пребывают в некоем абстрактном трёхмерном *пространстве событий*; пользователь смотрит на это пространство извне.



Пространство событий

Регионы могут генерировать и собирать объекты, называемые *событиями*. Эти события могут перемещаться в одном из двух направлений через пространство событий (т.е. либо к пользователю, либо от него). Перемещаясь сквозь пространство событий, события взаимодействуют с другими регионами – таким образом приложения взаимодействуют друг с другом. Процессом, поддерживающим эту простую архитектуру, является *менеджер Photon'a*.

Используя регионы и события, можно легко создать все службы, требуемые для системы управления окнами – оконные менеджеры, драйверы и приложения. И поскольку процессы,

регионами которых управляет менеджер Photon'a, не обязательно должны размещаться на том же компьютере, что и менеджер Photon'a, легко выполнить распределённые по сети приложения.

Регионы и события

Программы Photon'a используют два базовых объекта: регионы и события. Регионы – объекты стационарные, тогда как события перемещаются через пространство событий.

Регион – это *одиночная*, фиксированная прямоугольная область, которую программа перемещает в пространстве событий. Регион обладает атрибутами, которые определяют, как он взаимодействует с событиями.

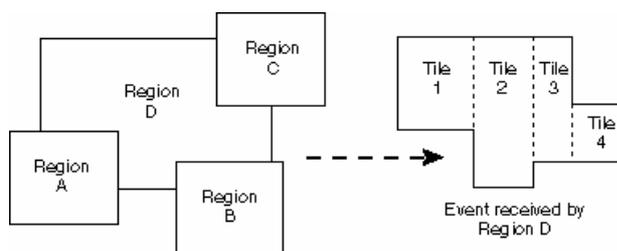
Некое событие представляет из себя *набор* неперекрываемых прямоугольников, которые могут генерироваться и собираться регионами в любом направлении пространства событий. Все события имеют какой-то присоединённый *тип*. Некоторые типы событий также владеют соответствующими данными.

События

Когда некое событие проходит сквозь пространство событий, его набор прямоугольников взаимодействует с регионами, размещёнными в пространстве событий другими приложениями. Когда такое происходит, менеджер Photon'a настраивает набор прямоугольников события в соответствии с атрибутами регионов, с которыми это событие взаимодействует.

Начальный набор прямоугольников

По умолчанию генерируемый событием начальный набор прямоугольников содержит один прямоугольник, размеры которого обычно являются размерами генерирующего региона. Когда событие проходит через пространство событий, его взаимодействия с другими регионами может привести к тому, что какие-то части этого прямоугольника могут быть удалены. Если это происходит, прямоугольник разделится на набор меньших по размеру прямоугольников, которые представляют оставшиеся части:



Набор прямоугольников события

Определённые типы событий (напр., нажатие кнопки), чтобы иметь размеры генерирующего диапазона, не требуют для себя начального набора прямоугольников. Для таких событий набор прямоугольников состоит из одного прямоугольника, размером в одну точку. Одноточечный набор прямоугольников называется *точечным источником*.

Накопленный набор прямоугольников

Набор прямоугольников "накопленных" событий состоит из прямоугольников, получившихся в результате взаимодействия события с предшествующими регионами в пространстве событий.

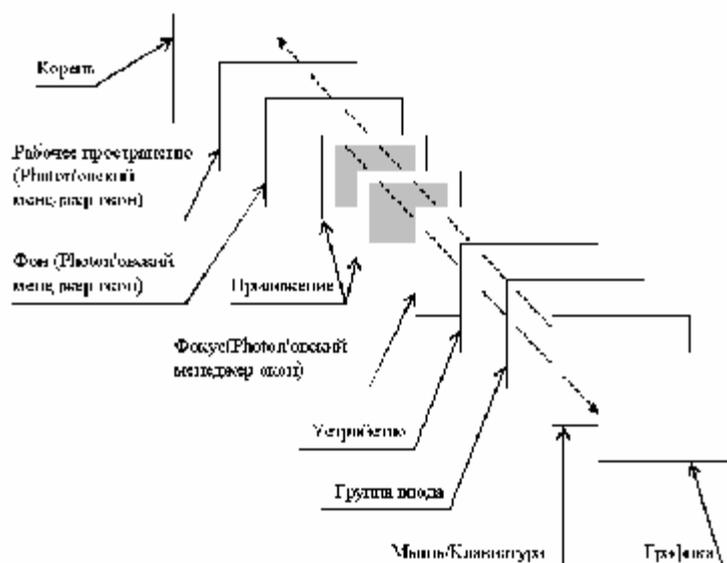
Если событие полностью перекрыто другими регионами, так что в результате набор не содержит прямоугольников, то такое событие пришло к концу.

☞ Список типов событий см. в разделе "Типы событий".

Регионы

Процесс может создавать или использовать любое количество регионов и может размещать их где угодно в пространстве событий. Более того, управляя размерами, местоположением и атрибутами региона (относительно других регионов пространства событий), процесс может использовать, модифицировать и удалять службы, предоставляемые другими регионами.

Photon использует серии регионов, выстроенных от корневого региона в конце пространства событий Photon'a к графическому региону в начале (вперед). События рисования стартуют в регионе приложения и движутся вперёд к графическому региону. События ввода стартуют на регионе Мышь/Клавиатура и перемещаются назад в направлении к корневому региону.



Расщеплённый образ регионов Photon'a

В файле <photon/PhT.h> определены следующие константы:

- Ph_DEV_RID – идентификатор региона устройств
- Ph_ROOT_RID – идентификатор корневого региона

Владелец регионов и менеджер Photon'a могут располагаться на разных компьютерах.

Регион имеет два атрибута, которые управляют тем, как с событиями обращаться, когда те пересекают регион:

- чувствительность
- непрозрачность

Вы можете устанавливать их независимо для каждого отдельного типа событий.

Чувствительность

Если регион чувствителен к какому-то конкретному типу события, то владелец региона собирает копии всех событий этого типа, которые пересекли регион. Если другие регионы чувствительны к этому же самому типу и событие с ними пересекается, то они тоже накапливают копии события – но с возможно другим набором прямоугольников.

Несмотря на то, что многие регионы могут собирать копии одного и того же события, набор прямоугольников для каждого события может быть скорректированным и поэтому будет уникальным для каждого региона, собирающего события. Набор прямоугольников отражает взаимодействие события с другими регионами в пространстве событий до того, как оно достигло накапливающего региона.

Если регион нечувствителен к некоему типу события, владелец региона никогда не будет накапливать этот тип события. Атрибут чувствительности не модифицирует набор прямоугольников события и не оказывает никакого влияния на способности события продолжать свой путь сквозь пространство событий.

Непрозрачность

Непрозрачные регионы блокируют дальнейшее прохождение набора прямоугольников события сквозь пространство событий. Атрибут непрозрачности управляет тем, корректируется ли набор прямоугольников события в результате взаимодействия с регионом.

Если регион непрозрачен для какого-то типа события, любое событие этого типа, пересекающее регион, получает набор прямоугольников события, откорректированный таким образом, что отсекается пересекаемым регионом. Эти изменения набора прямоугольников таковы, что они включают прямоугольники меньших размеров. Эти новые прямоугольники описывают части события, которые остались видимы для регионов, находящихся в пространстве событий позади этого региона.

Если какой-то регион *не является* непрозрачным для какого-то типа события, то для событий этого типа набор прямоугольников никогда не подвергается корректировке в результате пересечения с этим регионом.

Краткая сводка атрибутов

В следующей таблице суммируется, как атрибуты региона оказывают воздействие на пересекающие регион события:

Если регион является:	То событие является:	И набор для прямоугольников является:
Не чувствительным, не непрозрачным	Проигнорированным	Не подвергшимся воздействию
Не чувствительным, непрозрачным	Проигнорированным	Подвергшимся воздействию
Чувствительным, не непрозрачным	Собранным	Не подвергшимся воздействию
Чувствительным, непрозрачным	Собранным	Подвергшимся воздействию

Регистрация событий (event logging)

Размещая регион так, чтобы тот перекрывал всё пространство событий, процесс может взаимодействовать и модифицировать любые события, проходящие через регион. Если регион чувствителен ко всем событиям, но не является непрозрачным, он может прозрачно регистрировать все события.

Модификация событий

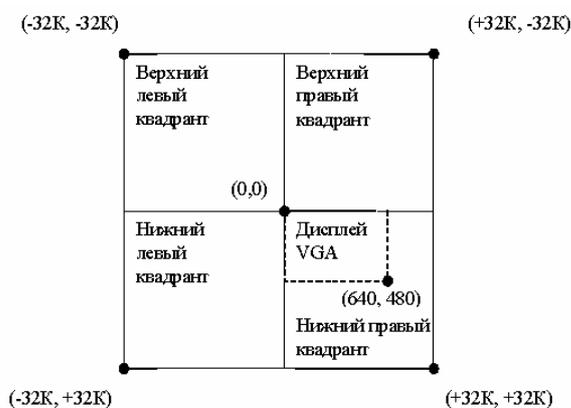
Если регион является чувствительным и непрозрачным, он может принять решение продолжить испускание уже модифицированной версии события. Например, регион может накапливать события мыши, выполнять в этих событиях распознавание рукописного текста, и затем генерировать эквивалентное событие клавиатуры.

Взаимосвязь родитель/потомок

Все регионы обладают взаимосвязями типа родитель/потомок. Регион потомка всегда располагается впереди родительского региона (т.е. ближе к пользователю), и его координаты являются относительными – задаются относительно родительского региона.

Координатное пространство Photon'a

Все регионы располагаются внутри координатного пространства Photon'a, имеющего следующие размеры:



Корневой регион

Самым дальним от пользователя всегда является особый регион, называемый *корневым регионом*. Все другие регионы в некотором роде произошли от него. Как только какое-то событие прошло от пользователя и достигло корневого региона, оно прекращает своё существование.

Размерами корневого региона являются размеры всего координатного пространства Photon'a. Как результат взаимосвязи родитель/потомок всех регионов, месторасположение любого региона в конечном счёте является относительным к размерам корневого региона.

☞ Регион может быть расположен в пространстве событий где угодно и тем не менее он имеет в качестве своего родителя корневой регион.

Типы событий

События генерируются по следующим причинам:

- нажата клавиша, информация о состоянии клавиатуры
- нажата и отпущена кнопка мыши
- движение мыши (с нажатой или ненажатой кнопкой)
- пересечение границы
- регион открылся или закрылся (exposed or covered)

- операции перетаскивания
- операция типа "тащи и бросай"
- функции прорисовки

Более полную информацию о типах событий см. в описании PhEvent_h "Справочника библиотечных функций Photon'a".

Как владельцы регионов уведомляются о событиях

Владельцы региона могут быть уведомлены о событиях менеджером Photon'a тремя различными способами:

- упорядоченным опросом (polling);
- синхронным уведомлением;
- асинхронным уведомлением.

Упорядоченный опрос

При упорядоченном опросе приложение вызывает некую функцию, которая запрашивает менеджер Photon'a с требованием немедленного ответа – либо с каким-то событием, либо с состоянием, указывающим, что никакое событие не доступно.

☞ Обычно Вам стоит избегать применения упорядоченного опроса, но при случае Вы можете найти этот способ полезным. Например, некое приложение, исполняющее на экране быструю мультипликацию, может выполнять упорядоченный опрос событий как часть своего потока событий прорисовки. Приложение может также использовать упорядоченный опрос, чтобы получить событие после асинхронного уведомления.

Синхронное уведомление

При синхронном уведомлении приложение вызывает функцию, которая запрашивает менеджер Photon'a с требованием немедленного ответа, если висит какое-то событие, либо ожидание перед выдачей ответа до тех пор, пока какое-либо событие не станет доступным.

При синхронном уведомлении приложение не может блокироваться на другом источнике, пока оно ожидает ответа от менеджера Photon'a. Вы можете посчитать такое поведение желательным в большинстве случаев, поскольку в результате приложение исполняется только когда становятся доступными нужные события. Но если по какой-то причине возможность блокировки на менеджере Photon'a является нежелательной, Вы можете рассмотреть возможность использования асинхронного уведомления.

Асинхронное уведомление

При асинхронном уведомлении приложение вызывает функцию, которая устанавливает метод уведомления (например, сигнал или импульс), который активизируется менеджером Photon'a, когда становится доступным событие нужного типа. Затем приложение может по опросу получать событие.

При асинхронном уведомлении приложение может блокироваться на нескольких источниках, включая процессы, не являющиеся приложением Photon'a.

Регион устройств

Владельцем региона устройств является менеджер Photon'a, который разделяет пространство событий на две части:

- *регион драйверов*, располагающийся на пользовательской стороне региона устройств;
- *регион приложений*, располагающийся на другой стороне региона устройств.

Менеджер Photon'a использует регион устройств для фокусирования событий указателя мыши и событий клавиатуры, а также для управления событиями перетаскивания.

Фокусировка указателя

Как и в других оконных системах, в Photon'e существует концепция *указателя* (т.е. экранного курсора). Этот указатель графически представлен на экране и отслеживает движения указательного устройства (напр., мыши или трекбола). Драйверы указательных устройств генерируют события, направляемые в сторону корневого региона.

Сгенерированное драйвером событие указателя является *нефокусированным*, или *необработанным*, до тех пор, пока оно не достигнет региона устройств, где менеджер Photon'a его перехватит и затем определит ему местоположение в пространстве координат Photon'a.

Определение этого местоположения – которое известно как *фокусировка* события – управляет тем, какие регионы будут накапливать событие. Затем менеджер Photon'a повторно сгенерирует событие из сфокусированного местоположения.

Поскольку Photon генерирует *сфокусированные* или *сфабрикованные* (cooked) события перемещения указателя в обоих направлениях от региона устройств, программы приложения могут в той же степени, что и программы драйвера, быть информированными о действиях указателя. Например, когда графический драйвер накапливает сфокусированные события указателя, он обновляет местоположение графического изображения указателя на экране.

Фокусировка клавиатуры

Драйвер клавиатуры похож на драйверы указательного устройства, за исключением того, что он генерирует события клавиатуры. Как и в случае событий указателя, события клавиатуры являются нефокусированными до тех пор, пока они не достигнут региона устройств, где менеджер Photon'a назначает им местоположение (т.е. фокусирует их) в координатном пространстве Photon'a. По умолчанию регион устройств устанавливает одно и то же местоположение фокуса и для событий клавиатуры, и для событий указателя. Поэтому регионы, расположенные непосредственно позади экранного указателя, будут накапливать сфокусированные события клавиатуры.

Оконный менеджер дополняет методы фокусировки клавиатуры. Более подробно см. раздел "Оконный менеджер Photon'a".

События перетаскивания

Приложение инициирует перетаскивание, генерируя событие перетаскивания для региона устройств. Как только это событие забирается регионом устройств, менеджер Photon'a берёт на себя заботу о взаимодействии с указателем (т.е. перетаскиваемым прямоугольником) до тех пор, пока операция перетаскивания не будет завершена. После завершения, регион устройств генерирует событие перетаскивания для приложения.

Событие "тащи и бросай"

Во время выполнения операции "тащи и бросай" генерируется серия событий, чтобы известить вовлечённые в эту операцию виджеты о состоянии операции. Некоторые из этих событий

генерируются для источника операции, остальные – для адресата этой операции. Более подробно см. главу "Ташить и бросать".

Драйверы фотона

В Photon'e драйверы не имеют существенных отличий от других приложений. Это просто программы, которые используют регионы и события конкретным образом, чтобы обеспечить свой сервис. В зависимости от своих функций драйвер может быть *драйвером ввода* или *драйвером вывода*.

Например, драйверы мыши и клавиатуры являются драйверами ввода, поскольку они генерируют события и являются источником воздействия аппаратного обеспечения. С другой стороны, графические драйверы являются драйверами вывода, потому что они накапливают события, которые становятся причиной того, что драйверы воздействуют на аппаратные устройства.

Драйверы ввода

Драйвер мыши

Драйвер мыши размещает регион на пользовательской стороне региона устройств. Он получает информацию от мыши как аппаратного устройства и строит необработанные события указателя Photon'a, которые затем генерируются в направлении корневого региона.

Драйвер клавиатуры

Драйвер клавиатуры также размещает регион на пользовательской стороне региона устройств. Драйвер получает информацию от клавиатурного аппаратного устройства и строит события клавиатуры Photon'a, которые затем генерируются в направлении корневого региона.

☞ Оконный менеджер добавляет принимаемый по умолчанию метод фокусировки, предоставляемый регионом устройств.

Поскольку Photon не делает предложений о том, какой тип клавиатуры используется, драйвер клавиатуры может запрашивать свои данные по событию от любого аппаратного устройства и даже от другого процесса.

Photon допускает несколько драйверов ввода и несколько драйверов вывода, присоединённых один к другому как некая *группа ввода*. Эта группа устройств будет Photon'ом ясно трактоваться для других групп ввода. Чтобы определить текущую группу ввода, вызовите функцию `PhInputGroup()`, передав ей текущее событие, если оно имеется.

Драйверы вывода

Графический драйвер

Графический драйвер располагает регион, чувствительный к событиям прорисовки, на пользовательской стороне региона устройств. Ввиду того, что драйвер собирает события прорисовки, он формирует на экране изображение с графической информацией. Поскольку набор прямоугольников, собирающих события, содержит только те области, которые нуждаются в обновлении, драйвер может оптимизировать их обновление. (Это особенно эффективно, если графическое аппаратное обеспечение может непосредственно обрабатывать списки отсечения).

☞ API прорисовки Photon'a накапливает запросы на прорисовку в пакеты, которые генерируются как одно событие прорисовки.

Несколько графических драйверов

Регион используемого графического драйвера может иметь размеры, представляющие область, размеры которой меньше размеров всего координатного пространства Photon'a. В результате несколько графических драйверов могут совместно использовать координатное пространство так, что каждый драйвер обрабатывает различные части этого пространства и отображает свои события на разных экранах. И поскольку владельцы регионов не обязаны быть на том же узле, что и менеджер Photon'a, эти графические драйверы могут отображать свои части координатного пространства на экранах, относящихся к другим компьютерам сети.

Драйверы, использующие отдельные регионы

С точки зрения приложения координатное пространство Photon'a всегда выглядит как одно, единое графическое пространство, несмотря на то, что Photon позволяет пользователю перетаскивать окна с одного физического экрана на другой.

Например, представим, что некий человек-оператор, работающий в среде управления предприятием, имеет портативный переносной компьютер. Если этот компьютер подсоединён к сети по беспроводной связи, оператор может войти в компьютер с полноэкранном управляющим приложением, и перетащить окно с этого экрана на экран портативного компьютера. Взяв с собой портативный компьютер, оператор теперь может пойти на производственный уровень предприятия и там и продолжить взаимодействовать с управляющим приложением для проверки и регулировки оборудования.

Драйверы, использующие перекрывающиеся регионы

В качестве другого примера, помимо наличия регионов драйверов с взаимоисключающими частями координатного пространства, мы можем рассмотреть драйверы, использующие *перекрывающиеся регионы*. Такой подход позволяет Вам сдублировать одни и те же события прорисовки на несколько экранов, что может быть идеальным для режима поддержки или обучения.

Инкапсуляция драйверов

Поскольку графические драйверы Photon'a в действительности являются обычными приложениями, они могут отображать графический вывод Photon'a внутри другой оконной системы (например, системы X Window). Драйвер Photon'a может также брать события клавиатуры и мыши, принятые из X Window, и повторно генерировать их внутри Photon'a, позволяя окну Photon'a в системе X Window быть полностью функциональным, как в отношении графического дисплея, так и ввода с клавиатуры или мыши.

Оконный менеджер Photon'a

Оконный менеджер является необязательным приложением Photon'a, которое управляет появлением и работой с меню, кнопками, линейками прокрутки и прочим подобным. Он обеспечивает для оконной системы свойство "видеть и осязать" (напр., Motif).

Оконный менеджер также управляет *рабочей областью*, дополнительными методами для фокусирования событий клавиатуры, и позволяет Вам отображать *фоновую плоскость*. Для

обеспечения всех этих служб оконный менеджер располагает в пространстве событий несколько регионов:

- Регионы оконных рамок
- Регион фокусировки
- Регион рабочей области
- Регион фона

Регионы оконных рамок

Большинство приложений в обеспечении пользователя средствами манипулирования размерами, позицией и состоянием (т.е. открыто/свёрнуто в иконку) этих приложений полагаются на оконную систему. Для того чтобы пользователь осуществлял эти действия, оконный менеджер размещает вокруг региона приложения рамку и затем располагает на этой рамке средства управления (напр., углы, брусок заголовка, кнопки). Мы говорим об этих средствах управления как о *оконных службах*.

Чтобы указать, что он может обеспечить оконные службы, оконный менеджер регистрируется у менеджера Photon'a. Когда приложение открывает окно, оконный менеджер устанавливает от своего имени два региона: регион рамки окна и регион приложения (или регион окна). Регион рамки окна чуть больше по размерам, чем регион окна и располагается прямо за ним.

Оконный менеджер использует регион рамки окна для его управления, тогда как приложение использует свой собственный регион. Но приложение не осведомлено об средствах управления окном. Если пользователь использует средства управления окном, чтобы переместить приложение, приложение уведомляется только о том, что его месторасположение изменилось. То же происходит при изменении размеров, сворачивании в иконку и прочая.

Регион фокусировки

Как обсуждалось ранее, регион устройств фокусирует события клавиатуры в регионы, расположенные непосредственно позади экранного указателя. Но при размещении региона, которым он владеет (т.е. *региона фокусировки*) сразу позади региона устройств, оконный менеджер перехватывает эти события клавиатуры, как будто они были сгенерированы из региона устройств, и выполняет альтернативный метод фокусировки.

Оконный менеджер может перенаправлять события клавиатуры на регионы, которые не расположены непосредственно позади экранного указателя. Например, он может фокусировать события в сторону последнего окна, на котором "щёлкнул" пользователь (т.е. *активное окно*). Оконный менеджер может направлять события клавиатуры на этот активный регион, даже если регион прикрыт другим регионом.

Регион рабочей области

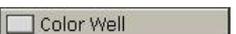
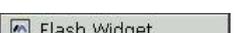
С точки зрения пользователя, рабочая область является пустым пространством, окружающим окна на экране. Оконный менеджер размещает *регион рабочей области* сразу перед корневым регионом, чтобы захватывать события до того, как они пройдут на корневой регион и таким образом исчезнут. Когда пользователь нажимает кнопку мыши и никакой регион не собирает это событие, оконный менеджер поднимает *меню рабочей области*, которое позволяет пользователю выбрать программу на исполнение.

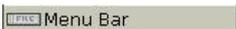
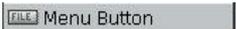
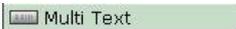
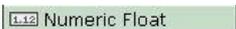
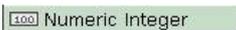
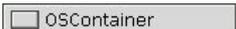
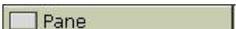
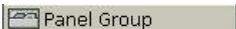
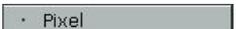
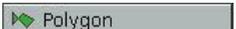
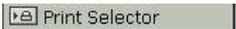
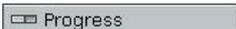
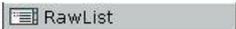
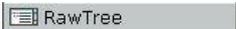
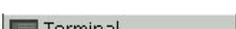
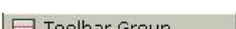
Регион фона

Пользователь часто любит иметь разукрашенную картинку в качестве фона, отображённую позади окон на экране. Чтобы отобразить такую побитовую карту, оконный менеджер располагает в пространстве событий *регион фона*.

ПРИЛОЖЕНИЕ 2. ОБЗОР ВИДЖЕТОВ

В данной таблице приведен список классов виджетов Photon'a и иконки, используемые в палитре виджетов PhAB. Для получения более полной информации о конкретных классах виджетов см. книгу "Справочник по виджетам".

Иконка PhAB	Класс	Описание
	PtArc	Эллиптический сектор
	PtBasic	Суперкласс – основа для большинства виджетов
	PtBezier	Кривая Безье
	PtBkgd	Подложка для непрозрачных образов, градиентов, картинок.
	PtButton	Кнопка (для инициирования действия)
	PtCalendar	Календарь
	PtClient	Суперкласс для клиентских виджетов (не присутствует в типичной установке)
	PtClock	Аналоговые, цифровые или часы из «светодиодов»
	PtColorPanel	Панель цветов
	PtColorPatch	Виджет предназначен для того, чтобы помочь в выборе параметров цвета
	PtColorSel	Суперкласс для виджетов, предоставляющих возможность выбора цвета (не присутствует в типичной установке)
	PtColorSelGroup	Набор виджетов для выбора цвета
	PtColorWell	Прямоугольный виджет, отображающий цвет и позволяющий его изменить
	PtComboBox	Текстовое поле с возможностью выбора значения из списка
	PtCompound	Суперкласс для смешанных виджетов (не присутствует в типичной установке)
	PtContainer	Суперкласс для виджетов-контейнеров (не присутствует в типичной установке)
	PtDisjoint	Суперкласс для «разрозненных» (состоящих из отдельных частей) виджетов (не присутствует в типичной установке)
	PtDivider	Виджет, разделяющий дочерние виджеты и позволяющий изменять расстояние между дочерними виджетами.
	PtEllipse	Эллипс
	PtFileSel	Виджет в виде дерева, позволяющий выбирать файлы и папки
	PtFlash	Контейнер, отображающий анимацию в формате Macromedia Flash 4
	PtFontSel	Виджет для выбора атрибутов шрифта
	PtGauge	Суперкласс для виджетов - ?? (не присутствует в типичной установке)
	PtGenList	Суперкласс для виджетов – списков (не присутствует в типичной установке)
	PtGenTree	Суперкласс для виджетов – «деревьев» (не присутствует в типичной установке)
	PtGraphic	Суперкласс для виджетов-изображений (не присутствует в типичной установке)
	PtGrid	Сетка
	PtGroup	Группа – для создания используйте команду «Сгруппировать вместе» (« Group together»)
	PtLabel	Ярлык в виде текста, изображения или образа
	PtLine	Прямая линия (один сегмент)
	PtList	Список текстовых полей

	PtMenu	Меню – для создания меню воспользуйтесь специальным модулем меню
	PtMenuBar	Панель меню, располагаемая в верхней части окна
	PtMenuButton	Кнопка, при нажатии на которую всплывает меню или подпункт меню
	PtMeter	Виджет для отображения количественных характеристик чего-либо
	PtMultitext	Многострочное текстовое поле с возможностью отображения шрифтов с различными атрибутами
	PtNumeric	Суперкласс для числовых полей (не присутствует в типичной установке)
	PtNumericFloat	Поле для ввода чисел с плавающей запятой
	PtNumericInteger	Поле для ввода целочисленных данных
	PtOnOffButton	Кнопка, имеющая два состояния
	PtOSContainer	Контейнер внеэкрannого контекста, удобен для смены образов и отображения анимации без эффекта мерцания.
	PtPane	Контейнер, управляющий дочерними виджетами
	PtPanelGroup	Контейнер, управляющий закладками
	PtPixel	Набор точек
	PtPolygon	Набор соединенных сегментов-линий
	PtPrintSel	Составной виджет, позволяющий настроить печать
	PtProgress	Индикатор процесса
	PtRaw	Виджет в котором вы можете использовать низкоуровневые функции рисования.
	PtRawList	«Чистый» список
	PtRawTree	«Чистоу» дерево
	PtRect	Прямоугольник
	PtRegion	Регион фотона – должен создаваться PtCreateWidget()
	PtScrollArea	Суперкласс для «прокручивающихся» виджетов (не присутствует в типичной установке)
	PtScrollBar	Полоса прокрутки
	PtScrollContainer	Окно для просмотра элементов большого размера («большой виртуальной поверхности»)
	PtSeparator	Разделитель
	PtServer	Виджет-сервер – должен создаваться PtCreateWidget()
	PtSlider	«Ползунок» - механизм ввода целочисленных данных из определенного диапазона
	PtTab	Закладка
	PtTerminal	Эмулятор терминала
	PtText	Однострочное текстовое поле
	PtTimer	Таймер
	PtToggleButton	Кнопка у которой два состояния
	PtToolbar	Суперкласс для виджетов-панелей инструментов
	PtToolbarGroup	Группа элементов панели инструментов
	PtTree	Иерархическое дерево
	PtTrend	Отображение соединенных точек, сдвигающихся от исходного положения в указанном направлении и на величину, определяемую сопоставленными данными

 Tty	PtTty	Устройство – терминал
 Web Client	PtWebClient	Виджет для отображения HTML страниц
	PtWidget	Суперкласс для любого виджета (не присутствует в типичной установке)
	PtWindow	Окно – для создания следует использовать соответствующий модуль

Приложение 3. Поддержка многоязычности Unicode

В этом приложении описывается, как Photon обрабатывает символы различных языков. Это включает:

- Широкие и многобайтовые символы
- Unicode
- UTF-8 кодирование
- Функции преобразования
- Другие кодировки
- Драйверы клавиатуры
- Формирующие последовательности Photon'a

Photon спроектирован так, чтобы обрабатывать символы различных языков. Следуя стандарту Unicode, Photon обеспечивает разработчиков возможностью создания приложений, которые могут с лёгкостью поддерживать основные языки и алфавиты мира.

Unicode сконструирован на наборе символов ASCII, но использует 32-битовое кодирование для поддержки полностью многоязычного текста. Он не требует эскейп-последовательностей или управляющих кодов при задании любого символа на любом языке. Заметьте, что кодировка Unicode пригодна для трактовки всех символов – будь то буквы алфавита, иероглифы или символы – совершенно одинаковым образом.

При проектировании драйвера клавиатуры и механизмов обработки символов мы опирались на расширение X11 клавиатуры и стандарты ISO 9995 и 10646-1.

Широкие и многобайтовые символы

ANSI C включает в себя следующие понятия:

<i>широкий символ</i>	Символ, представленный как значение типа <code>wchar_t</code> , которое обычно больше чем <code>char</code>
<i>многобайтовый символ</i>	Последовательность из одного или более байтов, представляющих символ, хранимый в массиве <code>char</code> . Количество байтов зависит от символа
<i>широкосимвольная строка</i>	Массив типа <code>wchar_t</code>
<i>многобайтовый символ</i>	Последовательность многобайтовых символов, хранящаяся в массиве <code>char</code>

Unicode

Unicode является 32-битовой схемой кодирования:

Он упаковывает большинство международных символов в широкосимвольном представлении (два байта на символ)

Коды меньше 128 определяют те же символы, что и стандарт ANSI.

Коды между 128 и 255 определяют те же символы, что и в наборе символов ISO 8859-1.

Существует область для частного использования в диапазоне от 0xE000 до 0xF7FF; Photon отображает её следующим образом:

Глифы	Диапазон
Неотображаемые клавиши	0xF000 – 0xF0FF
Курсорный шрифт	0xE900 – 0xE9FF

Значение символов Unicode см. в файле /usr/include/photon/PkKeyDef.h. Более подробно об Unicode см. на вебсайте консорциума Unicode по адресу www.unicode.org.

UTF-8 кодирование

Известный прежде как UTF-2, формат преобразования UTF-8 (для "8-битовой формы") спроектирован для использования данных в символах Unicode в 8-битовом окружении UNIX. Каждое значение Unicode кодируется как многобайтовая последовательность UTF-8.

Вот некоторые из основных особенностей UTF-8:

- В UTF-8 представление кодов менее 128 такое же, как в стандарте ASCII, так что любая ASCII строка является также корректной строкой UTF-8 и представлена теми же символами.
- Значения ASCII, напротив, не участвуют в UTF-8 преобразовании, обеспечивая тем самым полную совместимость с исторически сложившимися файловыми системами, в которых лексический анализ производится на уровне байтов ASCII кодировки.
- UTF-8 кодирует набор символов стандарта ISO 8859-1 как двухбайтовые последовательности.
- UTF-8 упрощает преобразование в и из текста в Unicode.
- Первый байт указывает число следующих байтов в многобайтовой последовательности, позволяя эффективно заранее выполнять лексический анализ.
- Эффективным является нахождение начала символа в произвольном месте побайтового потока, поскольку Вам надо чаще всего просмотреть четыре байта вперёд, чтобы найти легко распознаваемый иницирующий байт. Например,

$$isInitialByte = (byte \& 0xC0) != 0x80;$$
- UTF-8 является довольно компактным в смысле количества байтов, использующихся при кодировке.

В настоящее время кодировка такова:

- Для многобайтовых кодировок первый байт устанавливает 1 в числе битов высокого порядка, равным числу байтов, использованных в кодировании; бит после них устанавливается в 0. Например, двухбайтовая последовательность начинается с 110 в первом байте.
- Для всех последующих байтов многобайтной кодировки первые два бита – это 10. Оставшееся значение байта в многобайтовом кодировании всегда больше или равно 0x80.

В следующей таблице показана бинарная форма каждого байта кодирования и минимальное и максимальное значения символов, представленных 1-, 2-, 3-, и 4-байтовым кодированием:

Длина	Первый байт	Следующие байты	Минимальное значение	Максимальное значение
Один байт	0xxxxxxx	нет	0x0000	0x007F
Два байта	110xxxxx	10xxxxxx	0x0080	0x07FF
Три байта	1110xxxx	10xxxxxx	0x0800	0xFFFF
Четыре байта	11110xxx	10xxxxxx	0x10000	0x10FFFF

- Действительное содержание многобайтового кода (т.е. широкосимвольного кода) является соединением XX битов в коде. Двухбайтовый код с кодировкой 11011111 10000000 кодирует широкий символ 11111000000.
- Там, где возможен более чем один способ кодирования значения (как, например, 0), законным является только наикратчайший. Нулевой символ всегда является одним байтом.

Функции преобразования

В наших библиотеках языка C предполагается, что "широкий символ" является Unicode, и "мультибайт" – это UTF-8 в принятой по умолчанию локализации. Тип `wchar_t` определён как беззнаковый 32-битовый тип, и функции `wctomb()` и `mbtowc()` осуществляют UTF-8 кодирование в принятой по умолчанию локализации.

- ☞ Многобайтовые символы в библиотеке C являются UTF-8 в принимаемой по умолчанию локализации; в различных локализациях многобайтовые символы могут использовать различную кодировку.

Вы можете использовать следующие функции (описанные в "Справочнике библиотечных функций QNX") для преобразования между широкосимвольной и многобайтовой кодировками:

<code>mblen()</code>	Вычисляет длину многобайтовой строки в символах
<code>mbtowc()</code>	Преобразует многобайтовый символ в широкий символ
<code>mbstowcs()</code>	Преобразует многобайтовую строку в широкосимвольную строку

Библиотеки Photon'a используют многобайтовые UTF-8 строки символов: любая функция, обрабатывающая строки, должна быть в состоянии обработать действительную строку UTF-8, и функции, возвращающие строку, могут возвращать строку многобайтовых символов. Это также применяется к ресурсам виджета. Графические драйверы и сервер шрифтов предполагают, что все строки используют UTF-8.

Главная библиотека Photon'a `ph` предлагает следующие не-ANSI функции (описанные в "Справочнике библиотечных функций Photon'a") для работы с многобайтовыми UTF-8 и широкими символами:

<code>utf8len()</code>	Подсчитывает байты в UTF-8 символе
<code>utf8strblen()</code>	Находит количество UTF-8 символов во фрагменте строки
<code>utf8strchr()</code>	Ищет UTF-8 символ в строке
<code>utf8strichr()</code>	Ищет UTF-8 символ в строке, игнорируя регистр
<code>utf8strirchr()</code>	Ищет в обратном направлении UTF-8 символ в строке, игнорируя регистр
<code>utf8strlen()</code>	Находит длину строки UTF-8 символов
<code>utf8strnchr()</code>	Ищет UTF-8 символ во фрагменте строки
<code>utf8strncmp()</code>	Сравнивает фрагмент строки UTF-8 символов
<code>utf8strndup()</code>	Создаёт копию фрагмента строки UTF-8 символов
<code>utf8strnichr()</code>	Ищет UTF-8 символ во фрагменте строки, игнорируя регистр
<code>utf8strnlen()</code>	Находит количество байт, использованных для строки UTF-8 символов
<code>utf8strrchr()</code>	Ищет в обратном направлении UTF-8 символ в строке
<code>utf8towc()</code>	Преобразует UTF-8 символ в широкосимвольный код
<code>wctolower()</code>	Возвращает эквивалент широкого символа в нижнем регистре
<code>wctoutf8()</code>	Преобразует широкосимвольный код в UTF-8 символ

Эти функции определены в `<utf8.h>` (заметьте, это не `<photon/utf8.h>`) и используют UTF-8 кодировку, не принимая во внимание, каковой является текущая локализация. `UTF8_CUR_MAX` определяет максимальное число байтов в UTF-8 символе.

Другие кодировки

Если вашему приложению требуется работать с другими символьными кодировками, Вам надо преобразовать в и из UTF-8. Символьные наборы определены в файле `/usr/photon/translations/charsets` и включают:

- Big5 (Chinese)
- Cyrillic (KOI8-R)
- Japanese (EUC)
- Japanese (Shift – JIS)
- Korean (EUC)
- Western (ISO 8859-1)

Предлагаются следующие функции преобразования, описанные в "Справочнике библиотечных функций Photon'a":

PxTranslateFromUTF()	Преобразовывает символы из UTF-8
PxTranslateList()	Создаёт список всех поддерживаемых преобразований символов
PxTranslateSet()	Устанавливает новый символьный набор для преобразования
PxTranslateStateFromUTF()	Преобразует символы из UTF-8, используя внутренний буфер состояния
PxTranslateStateToUTF()	Преобразует символы в UTF-8, используя внутренний буфер состояния
PxTranslateToUTF()	Преобразует символы в UTF-8
PxTranslateUnknown()	Управляет тем, как обрабатывать неизвестные кодировки

☞ Эти функции поддерживаются только в статической форме в библиотеке `phexlib Photon'a`. Прототипы находятся в файле `<photon/PxProto.h>`.

Драйверы клавиатуры

Драйвер клавиатуры управляется таблично; он обрабатывает любую клавиатуру со 127 или немного меньшим количеством физических клавиш.

Нажатие клавиши сохраняется в структуре типа `PhKeyEvent_t` (описанной в "Справочнике библиотечных функций Photon'a").

Пример: текстовые виджеты

Текстовые виджеты используют область `key_sum` для отображаемых символов. Эти виджеты также проверяют её для обнаружения движения курсора. Например, если содержанием области является `Pk_Left`, курсор движется влево. `key_sum` имеет значение `Pk_Left` как для клавиши "курсор влево", так и для клавиши "курсор влево" на цифровой клавишной панели (предполагается, что NumLock выключено).

Слепые клавиши (dead keys) и скомпонованные последовательности

QNX поддерживает "слепые" клавиши и "скомпонованные" клавиатурные последовательности для генерации `key_sums`, которые отсутствуют на клавиатуре. Область `key_sum` применима только к нажатию клавиши, а не к её отпусканью, чтобы гарантировать, что Вы получите только один символ, а не два.

Например, если клавиатура имеет слепую клавишу ударения (например, `), и пользователь нажимает её следом за `e`, `Key_sum` принимает значение "e" со знаком ударения (è). Если клавиша

"e" не отпущена, и затем была нажата другая группа клавиш (или продолжена последовательность, или последовательность слепой клавиши), key_sums будет сохранять это в стеке до завершающего отпущения.

Если в ходе скомпонованной последовательности нажата неверная клавиша, драйверы клавиатуры генерируют key_sums для всех промежуточных клавиш, а не для действительного нажатия или отпущения.

Скомпонованные последовательности Photon'a

В Photon'e задействованы стандартные скомпонованные последовательности. Если Ваша клавиатура не включает в себя символы из стандартной таблицы кодов ASCII, Вы можете генерировать символ, используя скомпонованные последовательности. Например, символ ó может быть сгенерирован нажатием клавиши <Alt>, следом – клавиша <'>, и следом клавиша <o>.

☞ Это не сочетание клавиш; нажимайте и отпускайте каждую клавишу одна за другой.

Для генерирования букв с символом ударения могут использоваться следующие клавиши:

Клавиша	Ударение	Пример последовательности	Результат
'	сильное ударение	Alt ' o	ó
,	седиль	Alt , c	ç
^	циркумфлекс	Alt ^ o	ô
..	диарезис (трема)	Alt " o	ö
`	тупое ударение	Alt ` o	ò
/	наклонная	Alt / o	ø
~	тильда	Alt ~ n	ñ

Если на Вашей клавиатуре нет следующих символов, Вы можете создать их, нажав клавишу <Alt>, следом первую клавишу последовательности и следом вторую клавишу последовательности.

Символ	Описание	Значение Unicode	Последовательность
æ	Маленькая буква "æ" (лигатура)	E6	Alt e a
Æ	Заглавная буква "æ" (лигатура)	C6	Alt E A
Ð	Заглавная буква "eth"	D0	Alt D -
ð	Маленькая буква "eth"	F0	Alt d -
ß	Маленькая буква "острое s" (немецкая "заострённая s")	DF	Alt s s
µ	Знак микро	B5	Alt / U
ƿ	Маленькая буква "торн" (руническая "p")	FE	Alt h t
Ʀ	Большая буква "торн" (руническая "p")	DE	Alt H T
#	Знак номера	23	Alt + +
@	Коммерческое "at"	40	Alt A A
©	Знак копирайта	A9	Alt C 0 Alt C O Alt C o Alt c 0 Alt c O Alt c o
®	Знак зарегистрированной торговой марки	AE	Alt R O
[Левая квадратная скобка	5B	Alt ((
]	Правая квадратная скобка	5D	Alt))

{	Левая фигурная скобка	7B	Alt (-
}	Правая фигурная скобка	7D	Alt)-
»	Знак правых двойных угловых кавычек	BB	Alt >>
«	Знак левых двойных угловых кавычек	AB	Alt <<
^	Диакритическое ударение	5E	Alt > пробел
'	Апостроф	27	Alt ` пробел
`	Тупое ударение	60	Alt ` пробел
	Вертикальная черта	7C	Alt / ^ Alt V L Alt v l
\	Обратная наклонная черта	5C	Alt // Alt / <
~	Тильда	7E	Alt пробел
	Неразрывный пробел (no-break space)	A0	Alt пробел пробел
°	Знак градуса	B0	Alt 0 ^
¡	Знак перевернутого восклицательного знака	A1	Alt ! !
¿	Знак перевернутого вопросительного знака	BF	Alt ? ?
¢	Знак цента	A2	Alt C / Alt C Alt c / Alt c
#	Знак фунта [прим. пер. – знак # стоит в самом оригинале... Вообще-то, полагаю, должно быть "£"]	A3	Alt L - Alt L = Alt l - Alt l =
¤	Знак валюты	A4	Alt X 0 Alt X O Alt x 0 Alt X o Alt x O Alt x o
¥	Знак иены	A5	Alt Y - Alt Y = Alt y - Alt y =
	Разорванная (вертикальная) черта	A6	Alt ! ^ Alt V B Alt v b Alt
§	Знак параграфа	A7	Alt S ! Alt S 0 Alt S O Alt s ! Alt s 0 Alt s o
¨	Трема, или умляут	A8	Alt " "
·	Срединная точка	B7	Alt . .
ˆ	Седиль	B8	Alt , пробел
¬	Знак отрицания (not sign)	AC	Alt - ,
˜	Мягкий перенос	AD	Alt - -
˘	Макрон (знак долготы над гласным)	AF	Alt - ^ Alt _ ^ Alt _
±	Знак плюс-минус	B1	Alt + -
¹	Показатель степени 1	B9	Alt 1 ^ Alt S 1 Alt s 1
²	Показатель степени 2	B2	Alt 2 ^ Alt S 2 Alt s 2
³	Показатель степени 3	B3	Alt 3 ^ Alt S 3 Alt s 3
¶	Знак абзаца (знак параграфа)	B6	Alt P ! Alt p !

^a	Порядковый женский указатель (feminine ordinal indicator)	AA	Alt A - Alt a -
°	Порядковый мужской указатель (masculine ordinal indicator)	BA	Alt O _ Alt o _
$\frac{1}{4}$	Простая дробь одна четвёртая	BC	Alt 1 4
$\frac{1}{2}$	Простая дробь одна вторая	BD	Alt 1 2
$\frac{3}{4}$	Простая дробь три четверти	BE	Alt 3 4
/	Знак деления	F7	Alt - :
*	Знак умножения	D7	Alt x x

Приложение 4. Photon во встроенных системах

Это приложение включает в себя:

- Принимаемые допущения
- Введение
- Шаг 1. Экспорт переменной окружения `PHOTON_PATH`
- Шаг 2. Запуск сервера Photon'a
- Шаг 3. Запуск драйверов ввода (мыши, клавиатуры, чувствительного к касанию экрана, и проч.)
- Шаг 4. Запуск менеджера шрифтов
- Шаг 5. Переключение между графическими режимами
- Шаг 6. Запуск графического драйвера
- Шаг 7. Запуск оконного менеджера
- Шаг 8. Запуск Вашего приложения
- Пояснения
- Пример

Принимаемые допущения

1. Вы знакомы с Photon в десктопном окружении.
2. Вы понимаете процесс построения встроенных систем для QNX Neutrino. Более полную информацию см. в книге "Построение встроенных систем".
3. Вы будете использовать систему разработки QNX Neutrino для построения Вашей встроенной целевой системы, работающей в Photon'e.

Введение

Перед тем как Вы попытаетесь сконфигурировать Photon для запуска его в Вашей встроенной системе, мы рекомендуем Вам использовать предоставленную в этом приложении информацию для конструирования пробного встроенного окружения Photon на обычном PC.

В конце приложения мы включили файлы с примером, который Вы можете использовать. Если Ваш PC не имеет стандартной клавиатуры, msoft-совместимой мыши или видеокарты, поддерживаемой драйверами, используемыми в этих примерах, Вам понадобится модифицировать примеры так, чтобы они работали в Вашем окружении.

Когда Вы запускаете Photon в десктопном окружении, Вы бездумно набираете `ph`-сценарий, который выполняет за Вас всю работу. Этот сценарий:

- запускает Photon
- определяет Ваши устройства ввода
- запускает драйверы ввода
- определяет Ваше графическое устройство
- переключается в соответствующий графический режим
- запускает графический драйвер

- запускает менеджер шрифтов с требуемыми шрифтами
- запускает оконный менеджер
- запускает менеджер рабочего стола.

Как только Photon запущен, Вы запускаете нужные Вам приложения, используя либо оконный менеджер, либо менеджер рабочего стола.

Во встроенном окружении Вам понадобится выполнить все эти шаги самостоятельно – *вручную*. В этом есть много достоинств, поскольку позволяет Вам предопределить минимум файлов, необходимых Вашей системе, и точно задать, как будет запущен каждый драйвер и приложение.

Шаги по загрузке Photon'a

Вот основные шаги, необходимые для загрузки непосредственно в Photon с исполнением Вашего приложения (приложений).

1. Экспорт переменной окружения PHOTON_PATH.
2. Запуск сервера Photon'a.
3. Запуск драйверов ввода (мышь, клавиатура, чувствительный к касанию экран, прочая)
4. Запуск менеджера шрифтов
5. Переключение в графический режим
6. Запуск графического драйвера
7. Запуск оконного менеджера (необязательно)
8. Запуск Вашего приложения.

Каждый из этих шагов требует определённых файлов, которые должны быть установлены в Вашей целевой системе. При чёткой предопределённости того, какие графические устройства Вы имеете и какие шрифты требуются Вашему приложению, Вы можете иметь абсолютный минимум числа файлов (и требуемого дискового пространства).

Мы подробно пройдемся по всем этим шагам и обсудим, какие файлы требуется на каждом шаге. В конце Вы будете точно знать, какие файлы Photon'a Вам понадобятся, чтобы запустить на исполнение Ваше встроенное приложение.

Шаг 1. Экспорт переменной окружения PHOTON_PATH

Переменная окружения PHOTON_PATH предназначена для хранения директории с установленным Photon'ом. По умолчанию это директория /usr/photon. Предполагается, что в ней располагаются по меньшей мере следующие поддиректории:

bin	Исполняемые файлы Photon'a
font_repository	Файлы шрифтов Photon'a и конфигурационные файлы (ОС-независимые)
palette	Графические палитры (ОС-независимые)
translations	Photon'овские файлы перевода международных языков (ОС-независимые)

Вы должны установить переменную окружения PHOTON_PATH:

```
export PHOTON_PATH = /usr/photon
```

Шаг 2. Запуск сервера Photon'a

Если Вам не надо передавать серверу Photon'a никаких аргументов командной строки, он может быть запущен следующим образом:

```
Photon &
```

- ☞ Если в Вашем встраиваемом окружении имеется чувствительный к касанию экран или световое перо, Вы можете откорректировать значение ввода для событий указателя, задав опции **-D**, **-R** и **U**. Например, для исключения случайного изменения позиции из-за того, что нажатие пальцем захватывает больше чем один пиксель, задайте опцию **-U**. Более подробную информацию см. в описании Photon в книге "Справочник утилит QNX Neutrino".

Сервер должен располагаться в текущем **PATH** и путь должен быть определён до того, как команда будет запущена. В QNX Neutrino этим путём является `/usr/photon/bin`. Например:

```
export PATH=:/bin:/usr/bin:/usr/photon/bin
```

- ☞ Если загрузочный образ Вашей системы слишком велик из-за того, что Вы включили в неё Photon или другие исполняемые файлы, Вы можете смонтировать файловую систему, подождать её появления и затем загрузить исполняемые файлы из файловой системы на этапе загрузки. Более подробно см. описание утилиты `mkifs` в "Справочнике утилит QNX Neutrino".

Если Вы включаете в ваш загрузочный образ какой-либо исполняемый файл Photon'a, Вы должны также добавить в переменную окружения `MKIFS_PATH` путь `/usr/photon/bin`

Необходимые файлы

```
/usr/photon/bin/Photon
```

Шаг 3. Запуск драйверов ввода

Обычно для окружения в настольном компьютере Вы используете утилиту `inputtrap`, чтобы автоматически генерировать корректную командную строку и вызывать соответствующий драйвер `devi-*`. Например:

```
kbd fd -d/dev/kbd msoft
```

Обычно Вы запускаете `inputtrap`, поскольку наперёд не знаете, какой должна быть соответствующая командная строка.

Во встраиваемых системах устройства ввода часто располагаются по необычным адресам, не поддаются PnP-идентификации или просто не поддерживаются существующими `devi-*` драйверами. Кроме того, утилита `inputtrap` имеет склонность быть излишне большой и просто занимает драгоценную память в ограниченном окружении. Исходя из этих соображений, Вам обычно стоит задавать командную строку для драйвера `devi-*` вручную. (Вы можете временно проинсталлировать утилиту `inputtrap` и использовать её для создания корректной командной строки.)

Вы можете настроить драйверы ввода для своих нужд, используя набор инструментов Разработчика Драйверов Ввода (Input DDK). Например, Вы можете изменить размер базовой памяти или создать свой модуль для поддержки новых устройств.

Требующиеся файлы

Соответствующий драйвер `devi-*` в директории `/usr/photon/bin`.

Шаг 4. Запуск менеджера шрифтов

При построении встраиваемой системы Вам необходимо принять несколько решений об уровне поддержки шрифтов, включая то, какие шрифты Вам понадобятся, нужны ли Вам масштабируемые шрифты.

Первым шагом является принятие решения о том, какие шрифты Вам нужны:

- Очень вероятно, что Вам понадобится курсорный шрифт `phcursor.phf`.
- Если Ваша встраиваемая система включает в себя терминал `pterm`, Вам потребуются семейства шрифтов PC Terminal (`pcterm*.phf`), PC serif (`pcs*.phf`) и/или PC Sanserif (`pcss*.phf`). Вам, вероятно, понадобится также файл `$HOME/.photon/pterm.rc` или `$PHOTON_PATH/config/pterm.rc`, чтобы сконфигурировать шрифт терминала.
- Большинство базирующихся на виджетах приложений предполагают, что следующие алиасы:
 - TextFont
 - MenuFont
 - FixedFont
 - BallonFont
 - TitleFontопределены, будучи соответствующим образом отображены в файле отображения шрифтов (`fontmap`).
- Вэб-браузер требует следующие типы шрифтов:
 - шрифт основного тела (напр., PrimaSans BT , Dutch 801 Rm BT, прочая)
 - заголовочный шрифт (напр., Swis721 BT, прочая)
 - непропорциональный шрифт (напр., Courier 10 BT, PrimaSansMono BT, прочая).

Проверьте конфигурацию браузера, чтобы найти, какие шрифты ожидаются, и используйте эти шрифты, модифицируя конфигурацию так, чтобы отразить, что Вы инсталлировали, или используйте файл `fontmap`, чтобы отобразить это во время исполнения.

Вы можете отобразить или подставить имена шрифтов, используя файл `fontmap` или раздел `Mappings` утилиты `fontadmin`. Более полную информацию см. в "Справочнике утилит QNX Neutrino".

Конфигурирование шрифтов

Вы можете конфигурировать шрифты во встраиваемых системах сами, но проще использовать систему разработки, чтобы сконфигурировать шрифты для встраиваемой системы и присоединить данные о шрифтах и конфигурационные файлы к соответствующей директории образа построения Встраиваемой Файловой Системы (EFS).

Например, предположим, что корневая директория этого образа `–/usr/EKit/bsp/board/build/root` (реальный путь зависит от Вашего окружения системы разработки). Измените Вашу текущую директорию так, чтобы ею стала корневая директория Вашей встраиваемой поддиректории:

```
export EKIT_DIR=/usr/EKit/bsp/board/build/root
cd /usr/EKit/bsp/board/build/root
```

Затем создайте поддиректорию шрифтов для Вашей встраиваемой системы. Например:

```
mkdir -p my_dir /font_repository
```

Скопируйте необходимые файлы шрифтов в базу построения образа для сборки утилитой `mkeys`:

```
cp font_filename my_dir/font_repository (повторите для каждого шрифта)
cp /usr/photon/font_repository/font* my_dir/font_repository
cp /usr/photon/font_repository/phfont.ini my_dir/font_repository
```

```
mkfontdir -i $EKIT_DIR/my_dir/fontdir \  
-d $EKIT_DIR/my_dir/font_repository
```

ЗАМЕЧАНИЕ: утилита mkfontdir требует полный путь

В этом примере создана некая начальная конфигурация, обновлены файлы конфигурации шрифтов и скопированы все файлы шрифтов в заданную указанную директорию.

Чтобы изменить соответствие шрифтов, вызовите утилиту fontadmin:

```
fontadmin -c my_dir/font_repository -d my_dir/font_repository
```

Если Ваш, исполняющийся при запуске сценарий вызывает особый сервер шрифтов (напр., phfontphf или phfontFA), вы не получите пользы от предварительной обработки файла fontopt утилитой phfont; Вам надо вручную передать каждую опцию, содержащуюся в этом файле, в командную строку.

Вы можете отгестировать установку шрифтов до построения встраиваемого образа, перезапустив Ваш сервер шрифтов настольной системы, указав ему новую конфигурацию (вызвав утилиту phfont с опцией -d, задавая поддиректорию шрифтов из построенного образа). Чтобы вернуться к первоначальным установкам, просто выполните:

```
phfont &
```

Запуск сервера шрифтов

Если Вы сведущи в том, какие возможности сервера шрифтов Вам требуются (масштабируемые PFR – portable font resource – с помощью утилиты phfontpfr; или же простая поддержка побитовых отображений с помощью phfontphf), то Вы можете запустить непосредственно соответствующий сервер:

```
/usr/photon/bin/phfontphf &
```

Если файлов шрифтов нет в локальном образе, Вам надо задать директорию, содержащую эти файлы и конфигурацию, использовав опцию -d командной строки (либо для утилиты phfont, либо для реальных серверов phfontphf, phfontpfr и прочих):

```
/usr/photon/bin/phfontphf -d /fs/hd1-qnx4/my_dir/font_repository
```

Кроме того, если my_dir не является той, что и \$PHOTON_PATH во встроенной системе, для правильного местоположения необходимо указать опцию -d.

Доступные серверы располагаются в /usr/photon/bin и включают в себя:

phfontphf	Только шрифты побитовых образов
phfontpfr	Поддержка масштабируемых поточнобитовых ресурсов переносимых шрифтов (Scalable Bitstream PFR) и набора TrueType
phfontFF	Поддержка масштабируемых TrueType, Type1, Type2, Bitstream Speedo (только retail кодирование) и Bitstream Stroke шрифтов
phfontFA	Поддержка всего вышеперечисленного

Шаг 5. Переключение в графический режим

Обычно в окружении настольной системы Вы используете программу Photon'a crttrap – она может определить аппаратное обеспечение Вашего монитора и установить для него конфигурационный файл, где Вы можете выбрать предпочтительный графический режим и разрешение .

В окружении встроенных систем эта информация должна быть предопределена. Процесс переключения между графическими режимами и запуск графического драйвера должен быть выполнен вручную непосредственно заданием команд без использования удобных инструментов, таких как "перехватчики" (trappers) и приложения, конфигурирующие дисплей.

Во многих окружениях встраиваемых систем из соображений экономии отсутствует видеоBIOS. ВидеоBIOS ответственен за инициализацию микросхемы видеоконтроллера, и какие-либо другие установки, требуемые видеоподсистемами, такими как современные графические контроллеры, часто требуют тщательной настройки перед тем, как они могут быть использованы даже в режиме VGA.

В настольных системах в видеоBIOS'e вызывается код самотестирования при запуске BIOS и контроллер устанавливается в это время. Когда вызывается переключатель режимов, видеоконтролер находится в инициализируемом состоянии, так что переключателю режимов не требуется знать, как устанавливать контроллер. В дополнение к этому, многие переключатели режимов в настольных системах используют поддержку ядром виртуальной машины 8086 для исполнения кода в видеоBIOS для выполнения переключения режимов. Поддержка виртуальной машины 8086 доступна только для платформы x86.

Во встроенных системах, которые не используют BIOS, работа по установке видеоконтроллера ложится либо на код начального загрузчика программы (IPL code) (init_hw2), либо на переключатель режимов. Если система будет использоваться только в одном режиме, то код начального загрузчика обычно модифицируется таким образом, чтобы установить видеоконтроллер. Если предусмотрена поддержка более чем одного режима, используется специальный переключатель режима, который может полностью установить контроллер без использования ресурсов в видеоBIOS.

Установка карты в правильный режим

Есть несколько способов переключения карты в графический режим:

- Поместить в код начального загрузчика код, который переключает в графический режим во время загрузки. В системах x86, имеющих видеоBIOS, это может быть достигнуто исполнением программного прерывания для вызова видеоBIOS'a. Например:

```
mov ah, 0      ; установить режим
mov al, 12h    ; режим 0x12: 640x480, 16 цветов
int 10h
```

- Написать программу-переключатель режимов, которая может быть выполнена при запуске Photon'a
- Иметь динамически подключаемую библиотеку графических драйверов, выполняющую переключение режимов.

Если Вы решили переключать режимы с помощью DLL графических драйверов, возможно, что Вы решите использовать драйвер, предоставляемый либо QNX Software Systems, либо сторонними разработчиками.

Однако, если Вы хотите написать Вашу собственную программу переключения режимов, Вы, вероятно, найдёте полезным комплект разработчика графических драйверов (Graphics Driver Development Kit).

☞ В настоящее время все добавляемые нами драйверы требуют для корректной работы видеоBIOS. Некоторые драйверы для переключения видеорежимов выполняют вызовы в видеоBIOS. Эти драйверы работают только на системах платформы x86. Другие драйверы опираются на тот факт, что аппаратное обеспечение уже инициализировано (напр., во время загрузки через BIOS, загрузочное ROM или код начальной загрузки), тогда как третьи драйверы (напр., devg-banshee.so) способны сами инициализировать аппаратуру, но требуют информацию, хранящуюся в ROM BIOS'a. Если в Вашей целевой системе нет видеоBIOS'a, Вам, вероятно, понадобится индивидуально настроенный графический драйвер.

Шаг 6. Запуск графического драйвера

Графическая подсистема Photon'a запускается выполнением io-graphics. Вот несколько образцов вызова:

```
io-graphics -g640x480x8 -dldevg-vga.so -P/usr/photon/palette/vga4.pal
io-graphics -g1024x768x16 -dldevg-vesabios.so
io-graphics -g1024x768x16 -dldevg-rage.so -d0x1002,0x4755 -I0
```

Где опции означают:

- g** задаёт разрешение и глубину цвета выбранного видеорежима. Заметьте, что драйвер VGA пытается установить для io-graphics 8-битовую глубину цвета, даже когда устанавливает 4-битовый видеорежим из соображений производительности.
- dl** задаёт имя совместно используемого объекта графического драйвера, для того чтобы управлять графическим аппаратным обеспечением.
- d** требуется для драйверов, которые идентифицируют графическое аппаратное обеспечение по его идентификаторам PCI производителя и устройства.
- I** задаёт экземпляр PCI-устройства для подсоединения в случае, если в системе более одного графического устройства с одинаковыми идентификаторами производителя и устройства.
- P** задаёт для использования файл палитры; vga4.pal является палитрой, спроектированной для пользования в 16-цветном видеорежиме.

Более полную информацию об io-graphics см. в "Справочнике утилит QNX Neutrino".

Требуемые файлы

/usr/photon/bin/io-graphics	Запускает графическую подсистему
/lib/dll/devg-*	Графические драйверы аппаратного уровня
/usr/lib/libdisputil.so.1	Библиотека программ-утилит, используемых драйверами devg-*. Большинство графических драйверов скомпонованы с этой библиотекой.
/usr/lib/libffb.so.1	Библиотека программ растеризации, которые используют драйверы devg-*. Большинство графических драйверов скомпонованы с этой библиотекой.

Шаг 7. Запуск оконного менеджера

Этот шаг необязателен, если используется лишь одно приложение или если Ваша встроенная система не требует услуг, предоставляемых оконным менеджером.

В настоящий момент известно только одно неподдерживаемое свойство: множество сообщений Ph_WM_CLOSE к неотзывающемуся приложению не будут выставлять сигнал SIGHUP к этому процессу (до тех пор, пока операция в стиле канал-"убить сеть" (channel-"netkill" – style operation) не будет добавлена в ядро).

Файл конфигурирования рабочего пространства (\$HOME/.ph/wm/wm.cfg) может модифицироваться динамически с помощью утилиты pwmopts. Другим способом установки опций, не принятых по умолчанию, является установка через опции командной строки или через переменную окружения PHWMOPTS.

Персонализированное PWM-меню (\$HOME/.photon/wm/wm.cfg) оконного менеджера, если оно используется, может озадачить, если команды подаются как абсолютные маршруты имён (абсолютные командные имена должны быть совместимыми для разных систем, обеспечивая совместимость переменной среды PHOTON-PATH). Стандартное меню PWM вызывает все команды из \$PHOTON_PATH/bin. Заметьте, что для встроенных систем Вы можете полностью выключить это меню путём использования для pwm опции -W.

Необходимые файлы

/usr/photon/bin/pwm

Шаг 8. Запуск Вашего приложения

Если ваше приложение является единственным исполняющимся и не требует оконного менеджера, Вы можете линковать Ваше приложение как статическое – и Вам не нужна Photon'овская библиотека совместного доступа.

Если Вам требуется оконный менеджер или одновременно исполняются более одной программы Photon'a, лучше использовать Photon'овскую библиотеку совместного доступа.

Необходимые файлы

Файлы Вашего приложения.

Если Вашему приложению требуется библиотека совместного доступа –
/usr/photon/lib/libph.so

Пояснения

Замечено, что когда некоторые пользователи перемещают Photon в некую встраиваемую систему, они сталкиваются со следующими проблемами.

mkifs

По умолчанию утилита mkifs удаляет из исполняемых файлов имена ресурсов Photon'a. Чтобы избежать этого, задайте атрибут +raw для всех Photon'овских приложений.

Флэш-файловая система

На то, как Вы конфигурируете Photon, окажут влияние следующие особенности флэшевой файловой системы:

- Сжатие и скорость
Поскольку флэш-файловая система замедляется при работе со сжатыми файлами, Вы, вероятно, захотите хранить записи ресурсов в отдельном файле, а не включать их в конец бинарного файла. Чтобы это сделать, измените makefile таким образом, чтобы ресурсы привязывались к отдельному файлу. Например, измените следующую зависимость:

```
$(ABOBJ) $(MYOBJ)
$(LD) $(LDFLAGS) $(ABOBJ) $(MYOBJ) -M -o mine
usemsg mine ../Usemsg
phabbind mine $(ABMOD)
```

на:

```
$(ABOBJ) $(MYOBJ)
$(LD) $(LDFLAGS) $(ABOBJ) $(MYOBJ) -M -o mine
usemsg mine ../Usemsg
phabbind mine.res $(ABMOD)
```

Вам также понадобится экспортировать переменную пути AB_RESOVRD, если записи ресурсов находятся не в той же директории, что и исполняемые файлы. Это исключает из поиска директорию, содержащую исполняемые файлы.

- Поиск
Флэш-файловая система имеет ограничения на поиск и запись. Ваш код должен не допускать запись в середину файла.

☞ Процедуры, основанные на работе с конфигурационными файлами, с флэш-файловой системой не совместимы.

Графика

Во многих встроенных системах отсутствуют компоненты, обычные в настольных системах. Вот несколько тонкостей, которые можно ожидать:

BIOS ROMs	Поскольку многие переключатели режимов, поддерживаемые в Photon'e, требуют видеоBIOS для переключения графических режимов, возможно потребуются наличие BIOS'a на материнской плате. Справьтесь в QNX Software Systems по поводу того, доступна ли не BIOS-версия.
Текстовый режим	Photon не требует поддержки текстового режима, так что Вы можете убрать какие-либо установки, связанные с текстовым режимом.
Видеообласть	Память RAM можно сделать непрерывной, поскольку драйверы Photon'a не ограничены месторасположением видеообласти (напр., 0xA000). Вы можете разместить видеобуфер в любом месте памяти.

Различные проблемы

Вот ещё несколько замечаний по разным поводам:

Скорость процессора В определённых встроенных системах производительность процессора будет ниже, чем в настольных системах. Вам надо учитывать это при разработке приложений Photon'a для встраиваемой среды.

Прокрутка	Если при щелчке на жёлоб прокрутки область прокрутки пролистывается вниз более чем на одну страницу, попробуйте увеличить значение задержки повтора мыши в Photon'e. Например: Photon -D1000 &
Ввод	Вы можете установить параметры синхронизации передачи данных и для ввода и для сервера Photon'a. Путём снижения скорости генерирования событий мыши Вы можете уменьшить трафик через систему Photon'a. На медленных 386 и 486 платформах общепринятой практикой является снижение дросселирования ввода с 10 до 20 мс.
Phindows и Phditto	Если разрабатываемое Вами приложение требует поддержки удалённой диагностики с помощью Phindows или phditto, Вам может понадобиться установить phrelay, библиотеку визуализации и файл конфигурирования служб.

Пример

Давайте рассмотрим шаги, связанные со встраиванием Photon'a для использования в некоем встраиваемом устройстве. Нашей целью является построение системы Photon со следующими минимальными возможностями:

- Масштабируемые TrueType шрифты – наименьший доступный набор, включающий обычный шрифт, наклонный и жирный наклонный.
- Совершеннейший минимум, необходимый для работы графического драйвера под чипсет RageLT.
- Необязательные мышь/клавиатура – мы должны быть в состоянии запускать и останавливать эту службу по мере надобности.
- Необязательный оконный менеджер – мы должны быть в состоянии запускать и останавливать эту службу по мере надобности.

Встраивание Photon'a требует анализа ряда положений:

- Требуемые бинарные файлы
- Требуемые библиотеки (.so)
- Требуемые шрифты
- Требуемые серверы
- Размещение их всех вместе
- Полезные подсказки

Требуемые бинарные файлы

Первый шаг затрагивает проверку полной системы. Запустите Photon на Вашем компьютере. Прсмотрите вывод команды *pidin arg*. Это вывод в Gateway Laptop, выбирающий для показа только компоненты, относящиеся к Photon'у:

```
pidin ar
  pid Arguments
3620894 Photon
3665951 fontsleuth -d /usr/photon/font_repository
3727406 pwm
3657775 /usr/photon/bin/phfontFA -d /usr/photon/font_repository -j -s 300k
3698736 io-graphics -g1024x768x32 -dldevg-rage.so -I0 -d0x1002,0x4c42
3715121 devi-hirun kbd fd -d/dev/kbd ps2 kb -2
3772466 shelf
3809331 bkgdmgr
3809332 wmswitch
3809336 Xphoton -once
3809337 gtwm
```

Нам нужны только несколько программ:

- Photon
- phfontFA – см. обсуждение шрифтов ниже
- io-graphics
- pwm – только если нам нужно обслуживание оконного менеджера
- devi-hirun – только если нам нужны мышь или клавиатура (или чувствительный к прикосновению экран); см. раздел "Драйверы ввода (devi-*)" в сводке "Справочника утилит QNX Neutrino".

Для большинства встраиваемых систем другие компоненты являются совершенно необязательными:

font sleuth используется для автоматической установки шрифтов
 shelf Для "быстрого запуска" приложений. Она создаёт "полку" (по умолчанию по правой стороне экрана) для приложений, которые Вы можете запускать
 bkgdmgr Для рисования фоновой картинке экрана
 wmswitch Работает с pwm для обработки переключений между приложениями при нажатии Alt-Tab
 Xphoton Для запуска X-приложений
 gtwm Xphoton-овский оконный менеджер

Сохраните список аргументов для Вашей системы в файле. Он нам позднее понадобится.

Требующиеся библиотеки

В нашей системе нам требуются только следующие компоненты:

- Photon
- photonFA (или аналогичный менеджер шрифтов – см. обсуждение шрифтов ниже)
- io-graphics
- pwm
- devi-hirun

Посмотрим на вывод команды *piding mem*.

```

1048603  1 /photon/bin/Photon 10r RECEIVE          64K 120K 8192 (516K) *
         ldqnx.so.1 @b0300000          300K 12K
1302557  1 usr/photon/bin/pwm 10o RECEIVE          116K 56K 8192 (516K) *
         ldqnx.so.1 @b0300000          300K 12K
         libph.so.1 @b034e000        1220K 48K
         libphrender.so.1 @b048b000      232K 8192
1085470  1 hoton/bin/phfontFA 12r RECEIVE          284K 880K 12K (516K) *
         ldqnx.so.1 @b0300000          300K 12K
         /dev/mem @40100000 ( 0)      32K
1122335  1 io-graphics        12r REPLY            144K 148K 8192 (516K) *
         ldqnx.so.1 @b0300000          300K 12K
         libph.so.1 @b034e000        1220K 48K
         libphrender.so.1 @b048b000    232K 8192
         devg-rage.so @b04c7000           24K 4096
         libffb.so.1 @b04ce000         28K 4096
         libdisputil.so.1 @b04d6000         24K 4096
         /dev/mem @40100000 ( 0)      32K
         /dev/mem @40108000 ( 0)      4096
         /dev/mem @40109000 ( 4000)    64K
         /dev/mem @40119000 ( 0)      8192K
         /dev/mem @40919000 (fd7ff000) 4096
         /dev/mem @4091a000 ( 0)      2304K
1138720  1 o/x86/o/devi-hirun 15o RECEIVE          52K 24K 8192 (516K) *
1138720  2 o/x86/o/devi-hirun 10o REPLY            52K 24K 4096 (132K)
1138720  3 o/x86/o/devi-hirun 12o SIGWAITINFO       52K 24K 4096 (132K)
1138720  4 o/x86/o/devi-hirun 15o RECEIVE          52K 24K 4096 (132K)
         ldqnx.so.1 @b0300000          300K 12K
         libph.so.1 @b034e000        1220K 48K
         libphrender.so.1 @b048b000    232K 8192
    
```

Этот листинг говорит нам о каждой библиотеке, которая нам понадобится во встраиваемой системе. Лэптоп имеет видеочипсет Rage (devg-rage.so).

Таким образом, нам нужны следующие библиотеки (по меньшей мере):

- ldqnx.so.1
- libph.so.1
- libphrender.so.1
- devg-rage.so
- libffb.so.1
- libdisputil.so.1

Требуемые шрифты

Теперь давайте посмотрим на шрифты. Бывает, что приложение рассчитывает на какой-то определённый шрифт и обращается непосредственно к этому шрифту. В таком случае Вам необходимо явным образом включить каждый шрифт, необходимый приложению. Если Вы стандартизировали приложение в определённом наборе семейств/стилей шрифтов или Вам нет необходимости заботиться о том, какие именно шрифты у вас есть (до тех пор, пока их размеры Вас устраивают), то Вы можете урезать набор шрифтов и использовать один шрифт для замены нескольких других семейств шрифтов. Так, например, шрифт Times может использоваться в качестве замены Helvetica и Courier.

А теперь самое время построить сцену, на которой начнём тестирование нашей встроенной системы. Создадим под рутом поддиректорию /phembed. Внутри неё создадим поддиректории:

- phembed/bin
- phembed/lib
- phembed/font_repository

Теперь вернёмся к шрифтам. В нашем примере мы хотим использовать шрифт primasansbts TrueType почти во всех случаях. Для наших (необязательных) окон терминала rterm мы будем использовать шрифт фиксированной ширины rsterm. Также время от времени нам понадобится использовать мышь, так что мы подключим файл phcursor.phf.

Вот какие файлы нам требуются:

```
-rw-rw-r-- 1 root      root          707 Nov 29 15:20 fontdir
-rw-rw-r-- 1 root      root          104 Mar 20 2000 fonttext
-rw-rw-r-- 1 root      root           697 Nov 29 15:19 fontmap
-rw-rw-r-- 1 root      root        12393 Mar 20 2000 pcterml2.phf
-rw-rw-r-- 1 root      root        12905 Mar 20 2000 pcterml4.phf
-rw-rw-r-- 1 root      root        17437 Mar 20 2000 pcterml20.phf
-rw-rw-r-- 1 root      root         2868 Mar 20 2000 phcursor.phf
-rw-rw-r-- 1 root      root         75784 Mar 20 2000 tt2001m_.ttf
-rw-rw-r-- 1 root      root         77924 Mar 20 2000 tt2002m_.ttf
-rw-rw-r-- 1 root      root         71200 Mar 20 2000 tt2003m_.ttf
-rw-rw-r-- 1 root      root         82452 Mar 20 2000 tt2004m_.ttf
-rw-rw-r-- 1 root      root         56156 Mar 20 2000 tt2009m_.ttf
-rw-rw-r-- 1 root      root         58748 Mar 20 2000 tt2011m_.ttf
```

Скопируйте эти файлы из /usr/photon/font_repository в нашу /phembed/font_repository, затем измените директории на /phembed/font_repository.

Нам надо модифицировать файлы fontdir, fontmap и fonttext, чтобы отразить шрифты и отображения (mappings), которые нужны нам в нашей встроенной системе.

Вот модифицированный файл fontdir:

```
;
; fontdir config file, Tue Jan 18 15:34:42 2000
;
phcursor,.phf,Photon Cursor,0,,E900-E921,Np,32x32,3K
primasansbts.0@tt2001m_.ttf,PrimaSans BT,0,,0020-F002,Mlp,133x129,75K
primasansbtsi.0@tt2002m_.ttf,PrimaSans BT,0,I,0020-F002,Mlp,134x129,77K
```

```

primasansbtsb.0@tt2003m .ttf,PrimaSans BT,0,B,0020-F002,M1p,143x130,70K
primasansbtsbi.0@tt2004m .ttf,PrimaSans BT,0,BI,0020-F002,M1p,145x129,81K
primasansmonobts.0@tt2009m .ttf,PrimaSansMono BT,0,,0020-F002,M1f,60x129,55K
primasansmonobtsb.0@tt2011m .ttf,PrimaSansMono BT,0,B,0020-F002,M1f,60x130,58K
pcterm12,.phf,PC Terminal,12,,0000-00FF,Nf,6x12,13K
pcterm14,.phf,PC Terminal,14,,0000-00FF,Nf,8x14,13K
pcterm20,.phf,PC Terminal,20,,0000-00FF,Nf,10x19,18K
    
```

Как можно видеть из вышеприведенного списка:

- tt2001 – обычный шрифт (пропорциональный)
- tt2002 – курсив
- tt2003 – жирный
- tt2004 – жирный курсив
- tt2009 – одномерный обычный шрифт (не-пропорциональный)
- tt20011 – жирный одномерный

Шрифты pcterm12/14/20 предназначены для наших сессий pterm, и phcursor – для изображений курсора мыши. Вот модифицированный файл fontmap:

```

;
; fontmap config file, Tue Jan 18 15:34:42 2000
;
BalloonFont = primasansbts
FixedFont = primasansmonobts
HeadingFont = primasansbts
MenuFont = primasansbts
MessageFont = primasansbts
TextFont = primasansbts
TitleFont = primasansbts
Helvetica = primasansbts
Verdana = primasansbts
monospace = primasansmonobts
sans-serif = primasansbts
serif = primasansbts
web = primasansbts
arial = primasansbts
    
```

term = pcterm

```

geneva = primasansbts
monaco = primasansbts
ny = primasansbts
courier = primasansmonobts
dutch = primasansbts
swiss = primasansbts
times = primasansbts
wingbats = primasansbts
helv = primasansbts
ncen = primasansbts
time = primasansbts
? = primasansmonobts
    
```

Вы можете ещё ужать этот файл, отобразив все незаданные шрифты в шрифт "?". Это зависит от того, какие шрифты Вы хотите иметь фиксированной ширины, а какие – пропорциональными. Вот модифицированный файл fontext:

```

;
; fontext config file, Mon Dec 13 15:36:21 1999
;
+normal = primasansbts, primasansmonobts, phcursor
    
```

Службы шрифтов

Существует несколько доступных служб шрифтов:

- phfontFA Поддерживает все шрифты (.phf и .ttf), включая шрифты .pfr, созданные по старой технологии масштабируемых шрифтов.
- phfontFF Поддерживает все шрифты (.phf и .ttf), но не шрифты .pfr.

Если Вам не требуется из соображений работы с унаследованным ПО поддержка .pfr, мы рекомендуем использовать службу phfontFF.

Сборка всего этого в единое целое

А теперь соберём все эти кусочки вместе и построим простой сценарий, который будет запускать наш встроенный Photon. Вы должны уже были создать следующие директории:

- /phembed/bin
- /phembed/lib
- /phembed/font_repository

Скопируйте в /phembed/bin необходимые бинарники:

```
cp /usr/photon/bin/Photon /phembed/bin
cp /usr/photon/bin/phfontFF /phembed/bin
cp /usr/photon/bin/io-graphics /phembed/bin
cp /usr/photon/bin/devi-hirun /phembed/bin
cp /usr/photon/bin/pwm /phembed/bin
```

Скопируйте в /phembed/lib необходимые библиотеки:

```
cp /usr/lib/libph.so.1 /phembed/lib
cp /usr/lib/libphrender.so.1 /phembed/lib
cp /usr/lib/libffb.so.1 /phembed/lib
cp /usr/lib/libdisputil.so.1 /phembed/lib
cp /lib/dll/devg-rage.so /phembed/lib
```

Нам понадобится ещё одна библиотека. Если Вы хотите запускать приложения, разработанные в PhAB, которые требуют libAp.so.1, то Вам эта библиотека и понадобится. Мы рекомендуем сделать её доступной:

```
cp /lib/libAp.so.1 /phembed/lib
```

Нам также надо создать линки в директории /phembed/lib. Это нужно для того, чтобы приложения, которые вместо .so.1 смотрят в .so, работали нормально. Выполните следующее:

```
cd /phembed/lib
```

```
ln -s libAp.so.1 libAp.so
ln -s libph.so.1 libph.so
ln -s libphrender.so.1 libphrender.so
ln -s libffb.so.1 libffb.so
ln -s libdisputil.so.1 libdisputil.so
```

Теперь рассмотрим графические драйверы. В приведенном выше примере мы запускали драйвер devg-rage.so со следующими опциями:

```
dldevg-rage.so -IO -d0x1002,0x4c42
```

(Мы говорили об этом при обсуждении выхода pidin arg).

Если у Вас имеется другой графический драйвер, Вы должны его скопировать в /phembed/lib.

Теперь мы можем снова взглянуть на шрифты. У Вас имеется директория /phembed/font_repository, заполненная нужными Вам файлами .ttf и содержащая файлы fontmap, fontdir и fonttext, модифицированные так, как это описано выше.

Теперь у нас есть все кусочки, необходимые нам для того, чтобы попытаться собрать нашу встроенную систему Photon. Простейшим способом попробовать это выполнить – использовать вторую машину, соединённую с первой по telnet или нуль-модемному последовательному кабелю.

Если Вы используете нуль-модемный кабель, Вы можете запустить оболочку shell на Вашей терминальной машине, набрав `-t /dev/ser1 Ksh` на машинке с QNX Neutrino. Если скорость передачи (baud rate) и управление потоком на последовательных портах согласованы, на Вашем терминале появится приглашение оболочки (#). Мы предполагаем, что вы добились работы этого соединения и что у вас доступно приглашение оболочки на каком-то терминале, который связан с Вашей машиной QNX Neutrino.

Полезно было бы иметь для запуска Photon'a какой-то скрипт – основной сценарий. Вот основной сценарий, который запускает Photon, использующийся в приведенном выше примере:

```
cd /phembed/bin
```

```
./Photon &  
on -w /dev/photon -W10
```

```
./phfontFF -d /phembed/font_repository -c 20K -j -s 50K -F 10 -S 50 &  
on -w /dev/phfont -W10
```

```
./io-graphics -g1024x768x32 -dldevg-rage.so -I0 -d0x1002,0x4c42 &
```

```
/usr/photon/bin/phcalc -x100 -y100 &  
/usr/photon/bin/phcalc -x300 -y100 &
```

Скопируйте этот скрипт в файл `/phembed/ph-start` и сделайте его исполняемым (`chmod a+x ph-start`).

Обратите внимание в приведенном выше сценарии на следующее:

- Мы установили экспортируемые переменные так, чтобы они отражали нашу встроенную среду.
- Мы установили некоторые опции программы `phfontFF`, чтобы ограничить использование памяти. Для подробностей см. описание использования `phfontFF` (напр., `use /usr/photon/bin/phfontFF`).
- Графический драйвер запускается именно так, как мы увидели в опциях, напечатанных командой `pidin arg`. Если у Вас другой графический драйвер с другими опциями, Вам надо изменить эту строку.
- Мы запускаем несколько копий программы-калькулятора `phcalc`, просто чтобы показать, что Photon запущен.

До тех пор, пока не доступно имя требуемого устройства, для прекращения сценария мы используем команду `on`. Более подробно см. "Справочник утилит QNX Neutrino".

Если Вы всё выполнили удачно, Вы можете (из Вашего терминала) запустить основной сценарий `ph-start` и увидеть Photon с двумя окнами калькуляторов бок о бок на экране. Убедитесь, что Вы вышли из какой-либо сессии Photon'a перед тем, как запустить сценарий `ph-start`.

Полезные советы

Вот несколько полезных советов относительно отладки Вашей встроенной системы Photon:

- По умолчанию оконный менеджер `rwm` не запускается, так что у Ваших приложений нет никаких окаймляющих рамок, и вы не можете использовать мышшь для перемещения окон по экрану. Но ничего не мешает Вам запускать и останавливать `rwm` "на лету". Например, после запуска основного сценария `ph-start` наберите в терминальном окне `rwm &`
- По умолчанию не запускается драйвер мыши/клавиатуры. Он также может быть запущен и остановлен "на лету". Например, в нашей встроенной системе драйвер `devi` запускается следующей командной строкой:

```
/usr/photon/bin/devi-hirum kbd fd -d/dev/kbd ps2 kb -2 &
```

Мы можем выполнить эту команду по запросу оболочки с терминала и увидим появившийся указатель мыши, а также получим для использования клавиатуру. В полном смысле встроенной системе у Вас может не оказаться в наличии устройства устройства `/dev/kbd`, так что Вам понадобится-таки почитать документацию по семейству драйверов `devi-*`. Например, если у Вас имеется только PS2-порт:

```
devi-hirum kbd kb ps2 kb -2
```

- Вы можете запустить любую Photon'овскую программу по Вашему желанию и потом убить её. Например, если Вам нужно окно терминала, Вы могли бы запустить:

```
/usr/photon/bin/pterm &
```

И впоследствии завершить работу окна терминала.

Приложение 5. Использование PhAB под Microsoft Windows

В этом приложении описываются основные отличия между Windows'овской и натуральной QNX'овской версиями PhAB.

- Photon в одиночном окне
- Завершение PhAB
- Дополнительные опции
- Файловые имена с буквами в обоих регистрах
- DDD – Отладчик Отображения Данных (Data Display Debugger)
- Строка запуска отладчика
- Функциональность панели управления ресурсами
- Разработка индивидуальных виджетов и PhAB

Photon в одиночном окне

Как и натуральная QNX-версия PhAB, Windows-версия использует Photon и оконный менеджер Photon'a (pwm) для управления его окнами и диалогами. Основное отличие состоит в том что под Windows Photon выполняется внутри одного окна.

Когда Вы запускаете PhAB, он вначале запускает окно консоли, используемое только для сообщений состояния. Затем создаётся основное окно Photon'a. Все окна и диалоги PhAB появляются внутри этого основного окна, и все последующие приложения, запущенные PhAB, располагаются внутри этого окна. Заметьте, что Вы можете запустить несколько сессий PhAB внутри одного окна Photon'a.

Вы можете минимизировать окно приложения внутри окна Photon'a, но поскольку здесь не запущено приложение shelf, к правой кнопке мыши привязан список всех запущенных приложений, что позволяет Вам вытащить их из фонового режима. Чтобы это сделать, просто щёлкните правой кнопкой мыши на свободной области основного окна Photon'a и затем выберите то приложение, которое Вы хотите поднять из фонового режима.

Завершение PhAB

Когда Вы завершаете PhAB, тот пытается закрыть все компоненты Photon'a, если при этом в окне Photon'a не запущены какие-то другие приложения (такие, как другая сессия PhAB или редактор языка).

Если не все компоненты Photon'a завершаются автоматически или если Вы просто хотите принудительно закрыть всё в случае, когда у системы проблемы с запуском PhAB или компонентов Photon'a, Вы можете набрать команду

```
ph -kill
```

в ответ на приглашение команды Windows.

Дополнительные опции

Если Вы хотите задать опции командной строки для `rwmm` или сервера `photon`, Вы должны использовать следующие переменные окружения:

- `PWMOPTS`
- `PHOTONOPTS`

Чтобы установить эти переменные окружения, используйте ярлык "Environment" (в "system program" "Панели управления Windows"). Подробности об опциях командной строки для любой утилиты QNX см. в "Справочнике утилит QNX Neutrino".

Файловые имена с буквами в обоих регистрах

При переводе PhAB'овского проекта из QNX в Windows Вы можете столкнуться с проблемами, связанными с регистрами, в которых написаны имена файлов. QNX и UNIX-системы обращают внимание на регистры букв в именах файлов, тогда как Microsoft Windows – нет. Это приводит к следующим основным проблемам:

- Потеря оригинальных регистров букв в именах файлов при переводе. Одним из способов "добиться" этого – копирование всех файлов одновременно с использованием `FTR`. Например, файл `IntHandler.C` может на компьютере-адресате стать `INTHANDLER.C` или `inhandler.c`, что приведёт к ошибкам компиляции или линковки.
Одним из способов избежать этого – это использование под QNX утилиты `zip` для создания единого сжатого файла, и затем использование `Winzip` под Windows для распаковки `zip`-файла. `Winzip` учитывает регистры букв распаковываемых файлов.
- Потеря целых исходных файлов, чьи имена различаются только регистрами букв. Windows не хранит в одной директории несколько файлов, отличающихся только регистрами букв в именах. Здесь мы ничего не можем поделать, потому что это основа собственно Windows. Перед тем как переводить проекты из QNX в Windows, переименуйте такие файлы, так чтобы их имена отличались не только регистрами букв.

DDD – Отладчик Отображения Данных

DDD (Data Display Debugger) – это визуальный отладчик, предоставляющий графический интерфейс для GDB. Вы можете загрузить DDD для кросс-разработки QNX в Windows из экспериментального раздела на <http://qdn.qnx.com>.

- ☞ Проверьте, что загружаемая версия совпадает с текущей версией Вашего пакета кросс-разработки QNX.

Более полную информацию о DDD см. на <http://www.gnu.org/software/ddd>.

Если Вы установили DDD, Вы можете изменить предустановки ("Preferences") PhAB'a, так чтобы использовать DDD как отладчик по умолчанию (вместо версии `gdb`, работающей из командной строки). Вы можете использовать DDD и вне PhAB, просто набрав в командной строке Windows команду `ddd -debugger ntox86-gdb ...`

☞ DDD, равно как и PhAB, в настоящий момент не выполняется под Windows 95/98/Me. DDD является приложением X Windows System, что подразумевает, что у Вас должен быть запущен на компьютере X-сервер, для того чтобы отображать DDD. Когда DDD стартует, он использует любой X-сервер, который уже запущен. Если ничего не запущено, стартует простой принимаемый по умолчанию X-сервер.

Строка запуска отладчика

В диалоге "Build+Run" PhAB'a есть кнопка "Advanced Options", вызывающая диалог "Build Preferences" (более подробно см. главу "Генерация, компиляция кода и запуск на исполнение"). Здесь Вы можете определить команду отладчика, которую будет использовать PhAB. По умолчанию PhAB в Windows принимает команду:

```
gdb.bat -debugger nto$TPR-gdb --symbols
```

Она запускает команды пакетного файла gdb.bat. Если Вы установили DDD и хотите запускать его в качестве принимаемого по умолчанию отладчика, просто измените gdb.bat на ddd.bat и оставьте всё остальное как было.

PhAB автоматически устанавливает переменную окружения TPR перед вызовом команды отладки. Она содержит имя текущего целевого процессора, как он был определён при последнем исполнении построения проекта. В настоящий момент возможными значениями являются x86, ppc, mips, sh и arm. Наличие этой переменной в команде отладки приводит к автоматическому выбору нужного исполняемого файла отладчика для DDD.

Наконец, поскольку версия PhAB под Windows никогда не используется для хостинга самого себя, PhAB передаёт опцию --symbol в GDB по умолчанию. Это аналогично команде symbol отладчика GDB и делает символы загрузки gdb из PhAB исполняемыми, не делая его программой исполняемой, когда Вы используете команду run. Это позволяет Вам запустить исполняемый файл на удалённом компьютере. Вот команды инициализации из типичной сессии отладки после того, как был запущен DDD или GDB с использованием принятой по умолчанию командной строкой запуска:

```
(gdb) target qnx com1
(gdb) upload myprog
(gdb) run myprog
(gdb) break main
(gdb) continue
```

Мы здесь предполагаем, что мы подсоединены к целевой машине через последовательный порт com1 и что удалённый агент отладки rdebug на целевой машине уже запущен.

Функциональность панели управления ресурсами

Версия PhAB под Photon тормозит в сравнении с натуральной версией под QNX при заполнении панели управления ресурсами. Если панель управления ресурсами отображается и не перекрыта другой панелью управления, она обновляется каждый раз, когда Вы выбираете другой виджет.

Чтобы исключить эти нежелательные паузы и уменьшить время отклика при выборе виджета, Вам имеет смысл держать панель управления ресурсами скрытой под другой панелью управления до тех пор, пока Вам на самом деле она не понадобится.

Разработка индивидуальных виджетов и PhAB

Photon позволяет Вам создавать приложения, используя виджеты, которые Вы создали сами или получили от сторонних разработчиков. Вы можете встроить эти библиотеки "заказных" виджетов в своё приложение и запускать его на Вашей планируемой целевой машине.

Документация, описывающая написание исходного кода для разрабатываемых индивидуальных виджетов, получение их при работе Вашего приложения на целевой машине, и того, как добиться, чтобы PhAB распознавал Ваши индивидуальные виджеты, собрана в руководстве "Построение индивидуальных виджетов". Процесс, который делает все эти вещи, в сущности одинаков для всех платформ хост-машин.

PhAB умеет динамически подгружать индивидуальные виджеты и затем должным образом их отображать так, как Вы разработали Ваше приложение на хост-машине. До того как это стало возможным, Вы могли бы установить размеры и задать ресурсы и ответные реакции, но эти установки не будут в действительности иметь эффекта и отображаться до тех пор, пока Вы не запустите приложение на целевой машине.

Чтобы PhAB отображал индивидуальные виджеты корректно на хост-машине при разработке Вашего приложения, вам необходимо предпринять некоторые дополнительные шаги, которые зависят от платформы хост-машины. Вам понадобится перекомпилировать и слинковать исходный код индивидуальных виджетов под платформу и процессор хост-машины. Под ОС QNX это означает построение библиотек совместного доступа, которые PhAB динамически погружает во время исполнения приложения. Под Windows процесс слегка отличается и работает, используя статическую линковку.

Статическое линкование Ваших индивидуальных виджетов

Для следующей ниже процедуры мы предполагаем, что Вы уже выполнили шаги, которые не являются специфическими для платформы хост-машины. Вам нужно:

- Иметь исходный код индивидуальных виджетов
- Построить целевые библиотеки из Ваших исходников индивидуальных виджетов
- Добавить соответствующие входы в файлы определения палитры PhAB
- Создать изображение иконки и файлы ресурсов по умолчанию в директории шаблонов templates.

После того как Вы сделали всё вышперечисленное, выполните следующие шаги:

1. Загрузите и установите окружение разработчика Cygwin из www.cygwin.com. Это Unix-подобная система с открытым кодом, предлагающая основанное на gcc окружение разработки под Windows.
2. Напишите функцию C, действующую как таблица для просмотра индивидуальных виджетов PhAB'a для всех индивидуальных виджетов. Вы должны присвоить ей имя `get_custom_wgt()`, и она должна проверять передаваемое ей имя класса виджета и возвращать соответствующий указатель на класс виджета. Например:

```
#include <Pt.h>

extern PtWidgetClassRef_t *MyWidget;
```

```
PtWidgetClassRef_t **get_custom_wgt( const char *name ) {  
if (!strcmp(name, "MyWidget")) return &MyWidget;  
else return NULL;  
}
```

3. Откомпилируйте Вашу функцию индивидуального виджета и функцию таблицы просмотра, используя Cygwin'овский транслятор gcc:

```
gcc -c -D__LITTLEENDIAN__ -D__X86__ \  
-I /usr/lib/gcc-lib/i686-pc-cygwin/*/include \  
-I /usr/include \  
-I $QNX_HOST/usr/include -I$QNX_TARGET/usr/include \  
MyWidget.c get_custom_wgt.c
```

4. Сделайте резервную копию оригинального файла ab.exe, который имеется в поставке PhAB.
5. Слинкуйте Ваши объектные файлы индивидуальных виджетов, чтобы создать новую версию ab.exe:

```
ld $QNX_HOST/usr/photon/appbuilder/ab.o MyWidget.o \  
get_custom_wgt.o -o ab.exe -lcygwin -lkernel32
```

6. Замените имеющийся исполняющийся файл PhAB новым, Вами построенным. Вы должны выйти из PhAB, чтобы это сделать.

```
cp ab.exe $QNX_HOST/usr/photon/appbuilder/ab.exe
```

В следующий раз, когда Вы запустите PhAB после выполнения этих шагов, Вы сможете увидеть индивидуальные виджеты, корректно отображаемые, когда Вы с ними будете работать.

Глоссарий

- ❑ **accelerate**
Клавиша акселератор. - См. hotkey
- ❑ **activate**
Активация (приведение в действие). Виджет обычно *активирован*, когда Вы отпускаете кнопку мыши в то время, когда указываете на *взведенный (armed)* виджет.
- ❑ **activate window**
Активное окно. Окно, имеющее в данный момент *фокус (focus)*
- ❑ **anchor offset**
Смещение привязки. Расстояние между краями некоторого виджета и родительского виджета, к которому он *привязан (anchored)*.
- ❑ **anchor**
Привязка. Механизм ограничения, использующийся для управления тем, что происходит с виджетом, когда его родитель расширяется или сужается. Например, панель, привязанная к сторонам окна, расширяется или сужается, когда изменяются размеры окна.
- ❑ **application region**
Регион приложения. *Регион (region)*, принадлежащий приложению Photon'a (в противоположность системным процессам Photon'a, таким как оконный менеджер, графические драйверы, прочая). Регион приложения обычно размещается позади *региона устройств (device region)*. Также называется *регионом окна (window region)*.
- ❑ **argument list**
Список аргументов. Некий массив типа `PtArg_t`, используемый при установке и получении ресурсов виджета.
- ❑ **arm**
Взведение (приведение в состояние готовности). Виджет обычно *взводится (armed)*, когда Вы нажимаете кнопку мыши в момент, когда указатель мыши находится на виджете.
- ❑ **backdrop**
Фон. Изображение, которое отображается в качестве фона на Вашем экране.
- ❑ **backdrop regin**
Регион фона. Регион, размещённый позади всех окон для отображения фонового изображения.
- ❑ **balloon**
Всплывающая подсказка. Маленький прямоугольник, который всплывает для того, чтобы охарактеризовать или объяснить часть пользовательского интерфейса. Всплывающая подсказка отображается, когда указатель мыши задерживается над виджетом.
- ❑ **bitmap**
Побитовый образ. Цветная картинка, состоящая из одной или нескольких *побитовых плоскостей (bitplanes)*
- ❑ **bitplane**
Побитовая плоскость. Массив битов, представляющих в *побитовом образе (bitmap)* пиксели одного цвета.
- ❑ **blit**
Блитирование. Операция, перемещающая некую область графического контекста (напр., экран) в другую область с таким же или иным контекстом.
- ❑ **callback**
Ответная реакция. *Функция ответной реакции (callback function)* или *ресурс ответной реакции (callback resource)*.

- **callback function**
Функция ответной реакции. Код, соединяющий некий пользовательский интерфейс приложения с кодом приложения. Например, ответная реакция вызывается при нажатии на кнопку.
- **callback resource**
Ресурс ответной реакции. *Ресурс (resource)*, который определяет список функций и их клиентские данные, вызываемых при совершении определённого действия.
- **canvas**
Холст. Часть виджета, используемая под прорисовку. Для PtWidget это область внутри границ виджета. Для PtBasic и его потомков полотнищем является область границ виджета и *окаймления (margins)*. Другие виджеты, такие как PtLabel, могут определять дополнительные окаймления.
- **class**
Класс. См. *класс виджета (class widget)*.
- **class hierarchy**
Иерархия классов. Взаимосвязи между всеми классами виджетов.
- **client data**
Данные клиента. Какие-либо произвольные данные, которые нужны приложению для выполнения функции ответной реакции.
- **clipping list**
Список отсечений. Некий массив прямоугольников, используемый для ограничения вывода в определённые области.
- **clipping rectagle**
Прямоугольник отсечения. Прямоугольник, использующийся для ограничения вывода в определённую область.
- **CMY value**
Значение CMY. Цвет, выраженный в уровнях голубого, сиреневого и жёлтого (C[yan]M[agenta]Y[ellow])
- **CMYK value**
Значение CMYK. Цвет, выраженный в уровнях голубого, сиреневого, жёлтого и чёрного.
- **code-type link callback**
Связанная ответная реакция кодового типа. В приложении PhAB – некая функция, вызываемая при вызове списка ответных реакций.
- **color depth**
Глубина цвета. Число битов на пиксель для экрана или попиксельного отображения.
- **Common User Access**
Общепользовательский доступ. См. CUA.
- **compose sequence**
Составная (формируемая) последовательность. Последовательность нажатий клавиш, которая может использоваться для набора символов, которых может не оказаться на клавиатуре.
- **console**
Консоль. Один из девяти виртуальных экранов *рабочего стола (desktop)*. Также называется *рабочим пространством (workspace)*.
- **consume**
Поглощение. Когда виджет обработал какое-то событие и взаимодействие других виджетов с этим событием не допускается, говорят, что первый виджет *поглотил (consumed)* событие.

- ❑ **container**
Контейнер. Виджет, который может иметь другие виджеты в качестве потомков, например, PtWindow, PtGroup и PtOSContainer.
- ❑ **cooked event**
Сотворённое событие. Событие нажатия клавиши или кнопки мыши, назначенное какой-то локации в пространстве событий Photon'a. Также называется *сфокусированным событием (focused event)*.
- ❑ **CUA Common User Access**
Общепользовательский доступ. Стандарт, определяющий, как Вы можете изменить фокус, используя клавиатуру.
- ❑ **cursor**
Курсор. Некий указатель позиции на экране, такой как указатель *мыши (pointer)* или указатель вставки в текстовой области.
- ❑ **damaged**
Повреждённость. Каждый раз, когда какой-то виджет требует перерисовки в окне (напр., виджет был изменён, перемещён или *реализован (realized)*), говорят, что виджет повреждён.
- ❑ **dead key**
Пассивная клавиша. Клавиша, которая, будучи нажатой, не производит символ, а инициализирует *составную последовательность (compose sequence)*.
- ❑ **default placement**
Размещение по умолчанию. Размещение региона, когда для него не заданы братья. Противоположность *заданному размещению (specific placement)*.
- ❑ **desktop**
Рабочий стол. Виртуальный экран, состоящий из девяти *консолей (consoles)* или *рабочих пространств (workspace)*.
- ❑ **device region**
Регион устройств. *Регион (region)*, расположенный в середине *пространства событий (event space)*, с *регионами приложений (application regions)* позади него и *регионами драйверов (drives regions)* перед ним (с точки зрения пользователя).
- ❑ **dialog module**
Модуль диалога. *Модуль (module)* PhAB, похожий на *модуль окна (window module)*, за исключением того, что модуль диалога может существовать для каждого процесса только в одном экземпляре.
- ❑ **direct-color**
Напрямую определённый цвет. Цветовая схема, в которой каждый пиксель представлен значением RGB. Противоположность цвету, *основанному на палитре (palette-based)*.
- ❑ **disjoint parent**
Отделённый родитель. *Отделённый виджет (disjoint widget)*, являющийся прародителем другого виджета.
- ❑ **disjoint widget**
Отделённый виджет. Виджет, который может существовать без родителя. Если отделённый виджет имеет родителя, он может существовать вне полотноща своего родителя. Например, PtWindow, PtMenu и PtRegion являются отделёнными виджетами, а PtButton, PtBkgd, PtRect – нет. Отделённый виджет является владельцем регионов, которые не являются потомками регионов его родителя. Любой набор отсеечения родителя отделённого виджета не применяется к отделённому виджету. Регионы отделённых виджетов чувствительны и непрозрачны к испускаемым событиям.
- ❑ **dithering**
Сглаживание. Процесс, при котором пиксели двух цветов комбинируются для создания текстуры или смешанного цвета.

- **draw context**
Контекст прорисовки. Структура, описывающая поток прорисовки. Принимаемый по умолчанию контекст прорисовки генерирует события прорисовки для графических драйверов. *Контексты печати (print contexts)* и *контексты памяти (memory contexts)* являются типами контекстов прорисовки.
- **draw stream**
Поток прорисовки. Набор маркеров (токенов), которые отсылаются в событиях прорисовки и могут накапливаться движком визуализации, таким как графический драйвер.
- **driver region**
Регион драйверов. *Регион (region)*, созданный драйвером, обычно располагается перед *регионом устройств (device region)*.
- **encapsulation driver**
Инкапсулированный драйвер. Программа, которая отображает графический вывод Photon'a внутри другой оконной системы, такой как X Window System.
- **event**
Событие. Структура данных, которая представляет из себя некое взаимодействие между Вами и приложением или между приложениями. События проходят через пространство событий к Вам либо от Вас (т.е. в сторону *корневого региона (root region)*).
- **event compression**
Сжатие событий. Слияние событий таким образом, что приложение видит только их самые последние (поздние) значения. Приложению не приходится обрабатывать множество ненужных событий.
- **event handler**
Обработчик событий. Функция ответной реакции, которая позволяет приложению реагировать непосредственно на события Photon'a, такие как перетаскивание событий.
- **event mask**
Маска событий. Набор типов событий, которые представляют интерес для некоего *обработчика событий (event handler)*. Когда одно из них встречается, вызывается обработчик событий.
- **event space**
Пространство событий. Абстрактно, это трёхмерное пространство, содержащее регионы – от корневого региона сзади до графического региона впереди. Вы сидите вне пространства событий, глядя на него спереди. События проходят сквозь пространство событий в сторону корневого региона или в направлении к Вам.
- **exported subordinate child**
Экспортированный подчинённый потомок. Виджет, созданный виджетом контейнерного типа (в противоположность приложению), к чьим ресурсам Вы можете получить доступ только через его родителя.
- **exposure**
Дефект. Обычно происходит, когда *регион (region)* уничтожен, изменены его размеры или он перемещён. Событие дефекта посылается приложению, информируя его, когда содержание регионов приложения требует перерисовки.
- **extent**
Размер, занимаемое пространство. Прямоугольник, описывающий самые удалённые края виджета.
- **File Manager**
Файловый менеджер. Приложение Photon File Manager (PFM), использующееся для обслуживания и организации файлов и директорий.

- **focus**
Фокус. Виджет, имеющий *фокус*, будет получать любые события клавиатуры, накопленные его окном.
- **focus region**
Регион фокуса. Регион, помещённый непосредственно позади *региона устройств (device region)* оконным менеджером *Photon'a (Photon Window Manager)*, что позволяет ему перехватывать события клавиатуры и направлять их на *активное окно (active window)*.
- **focused event**
Сфокусированное событие. Событие клавиатуры или кнопки мыши, назначенное локации в пространстве событий *Photon'a*. Также называется *сотворённым событием (cooked event)*.
- **folder**
Папка. В файловом менеджере *Photon'a* – метафора директории.
- **GC**
Графический контекст. См. *графический контекст (graphics context)*.
- **geometry negotiation**
Согласование геометрии. Процесс определения расположения виджета и его потомков, который зависит от политики расположения виджетов, всех наборов размеров виджетов и размеров и желательных позиций каждого потомка виджета.
- **global header file**
Глобальный заголовочный файл. Заголовочный файл, который включается во весь код, генерируемый *PhAB'ом* для приложения. Глобальный заголовочный файл задаётся в *PhAB'овском* диалоге задания стартовой информации приложения (*Application Startup Information dialog*).
- **graphics driver**
Графический драйвер. Программа, которая размещает регион, чувствительный к событиям прорисовки, с пользовательской стороны региона устройств, накапливает события прорисовки и визуализирует на экране графическую информацию.
- **graphics context (GC)**
Графический контекст. Структура данных, определяющая характеристики примитивов, включая цвет переднего плана, цвет фона, ширину линий, отсечение, прочая.
- **Helpviewer**
Хэлпвьювер (да ну там, ну какой "Просмотровщик помощи???" Никто так не говорит... – Прим.пер.). Приложение *Photon'a* для просмотра он-лайн информации.
- **hotkey**
Горячая клавиша. Специальная клавиша или сочетание клавиш, которая вызывает некое действие (такое, как пункт меню) без фактического выбора виджета. Также называется *клавишей-акселератором (accelerator)*. Отличается от *клавиш быстрого доступа (keyboard short cut)*.
- **hotspot**
Горячая точка. Часть указателя мыши, соответствующая координатам, которые сообщаются указателю (например, точка пересечения перекрестья, или конец острия стрелки основного указателя).
- **HSB**
Цветовая модель "Оттенок – Насыщенность – Яркость" (Hue – Saturation – Brightness)
- **HSV**
Цветовая модель "Оттенок – Насыщенность – Значение" (Hue – Saturation – Value)
- **icon module**
Модуль иконки. Модуль *PhAB*, который связывает иконки с приложением.
- **image**
Образ. Прямоугольный массив значений цвета, в котором каждый элемент представляет один пиксель. См. также *напрямую определённый цвет (direct-color)* и *цвет, основанный на палитре (palette-based)*.

- ❑ **initalization function**
Функция инициализации. В приложении Photon'a это некая функция, которая вызывается до того, как будет создан какой-либо виджет.
- ❑ **input driver**
Драйвер ввода. Программа, которая генерирует и является источником событий клавиатуры и/или указателя мыши.
- ❑ **input group**
Группа ввода. Набор устройств ввода/вывода. Обычно имеется по одной группе ввода на пользователя.
- ❑ **input handler (or input-handling function)**
Обработчик ввода (или функция обработки ввода). Функция, которая подключена в главную петлю обработки событий для перехвата и обработки сообщений и *импульсов (pulses)*, посылаемых приложению другими процессами.
- ❑ **instance**
Экземпляр. Конкретный образец абстрактного класса; например, "Тузик" является экземпляром класса "Собака". В Photon'e экземпляр – это обычно экземпляр виджета; например, кнопка для нажатия является экземпляром виджетного класса QPushButton. Когда создаётся экземпляр виджета, для него определяются начальные значения его ресурсов.
- ❑ **instance name**
Имя экземпляра объекта. В PhAB – строка, идентифицирующая конкретный экземпляр виджета, так что Вы можете получить доступ к экземпляру из кода Вашего приложения.
- ❑ **instatiation**
Реализация экземпляра. Действие, создающее *экземпляр (instance)* виджетного класса в приложении.
- ❑ **internal link**
Внутреннее связывание. Механизм в PhAB, позволяющий разработчику получать доступ к модулю PhAB непосредственно из программного кода приложения.
- ❑ **Image Viewer**
Просмотрщик рисунков. Приложение Photon'a (pv), отображающее рисунки (образы).
- ❑ **Key modifier**
Модификатор клавиши. Флаг в событии клавиатуры, указывающий состояние соответствующей *клавиши-модификатора (modifier key)*, когда была нажата другая клавиша.
- ❑ **Keyboard driver**
Драйвер клавиатуры. Программа, получающая информацию от клавиатуры как аппаратного средства, выстраивающая Photon'овские события клавиатуры и генерирующая их в направлении корневого региона.
- ❑ **Keyboard shortcut**
Клавиша быстрого доступа. Клавиша, которая выбирает пункт меню. Клавиша быстрого выбора работает только тогда, когда меню отображено. Противоположность *"горячей" клавише (hotkey)*.
- ❑ **language database**
Языковая база данных. Файл, содержащий текстовые строки, используемые в приложении PhAB; языковая база данных упрощает создание многоязычных приложений с использованием языкового редактора PhAB'a.
- ❑ **link callback**
Связанная ответная реакция. Механизм, соединяющий различные части приложения PhAB. Например, связанная ответная реакция может вызываться, чтобы отобразить диалог при нажатии некой клавиши.
- ❑ **margin**
Граница. Область между рамкой виджета и его *полотнищем (canvas)*.

- ❑ **memory context**
Контекст памяти. *Контекст прорисовки (draw context)*, в котором Photon прорисовывает события, которые были направлены в память для дальнейшего отображения этого контекста на экране, в отличие от *контекста печати*, направляемого на принтер, или принимаемого по умолчанию контекста прорисовки, направляемого непосредственно (напрямую) на экран.
- ❑ **menu module**
Модуль меню. Модуль PhAB'a, используемый для создания меню.
- ❑ **method**
Метод. Функция, являющаяся внутренней по отношению к классу виджета и вызываемая при определенных условиях (например, прорисовка виджета). Методы обеспечиваются через указатели на функции в записях класса виджета.
- ❑ **modifier key**
Клавиша-модификатор. Клавиша (такая, как Shift, Alt или Ctrl), используемая для изменения смысла другой клавиши.
- ❑ **module**
Модуль. Некий объект в PhAB, который содержит виджеты приложения. Модули PhAB'a включают окна (windows), меню (menus), иконки (icons), картинки (pictures) и диалоги (dialogs).
- ❑ **module-type link callback**
Связанная ответная реакция модульного типа. Ответная реакция, которая выводит изображение некоего модуля PhAB.
- ❑ **mouse driver**
Драйвер мыши. Программа, которая получает информацию от аппаратного устройства позиционирования, выстраивает Photon'овские необработанные события указателя мыши и затем генерирует эти события в направлении корневого региона.
- ❑ **opaque**
Непрозрачность. Состояние региона в отношении к событиям. Если регион *непрозрачен (opaque)* к какому-то типу событий, то любое событие этого типа, которое интересует регион, имеет свой набор прямоугольников, установленный для вырезки интересующей области. Регион препятствует прохождению события через себя.
- ❑ **palette**
Палитра. Некий массив цветов. *Аппаратная палитра (hard palette)* имеется в аппаратном обеспечении; *программная палитра (soft palette)* – в программном обеспечении.
- ❑ **palette-based**
Основанный на палитре. Схема цветности, в которой каждый пиксель представлен индексом в палитре. Противоположность схеме *непосредственного цвета (direct-color)*.
- ❑ **PDP**
См. "*Нажать-перетащить-бросить*" (*Press-draw-release*).
- ❑ **PFM**
См. *Файловый менеджер Photon'a (Photon File manager)*.
- ❑ **PhAB**
Построитель приложений Photon'a (Photon Application Builder). Визуальное средство разработки, которое генерирует код, требующийся для реализации пользовательского интерфейса.
- ❑ **phditto**
Утилита, которая позволяет получить доступ к рабочему пространству Photon'a на удалённом узле. См. также *ditto*.
- ❑ **Phindows**
Photon в среде Windows. Приложение, позволяющее получить доступ к сессии Photon'a из среды Microsoft Windows.

- **PhinX**
Photon в среде X. Приложение, позволяющее получить доступ к сессии Photon'a из среды X Window System.
- **Photon File Manager (PFM)**
Файловый менеджер Photon'a. Приложение, использующееся для обслуживания и организации файлов и директорий.
- **Photon Manager or server**
Менеджер Photon'a или сервер. Программа, обслуживающая пространство событий Photon'a через управление регионами и событиями.
- **Photon Terminal**
Терминал Photon'a. Приложение (pterm), которое эмулирует символьный терминал в окне Photon'a.
- **Photon Window Manager (PWM)**
Оконный менеджер Photon'a. Приложение, которое управляет внешним видом оконных рамок и других объектов на экране. Например, оконный менеджер добавляет к окну приложения бруски изменения размеров рамки, брус заголовка и различные кнопки. Оконный менеджер также обеспечивает функционирование метода фокусирования событий клавиатуры.
- **picture module**
Модуль картинки. Модуль PhAB, содержащий некие систематизированные виджеты, который может быть изображён в другом виджете либо использоваться как база данных виджета.
- **pixmap**
Пиксельное отображение. *Побитовое отображение (bitmap) или образ (image).*
- **plane mask**
Маска плоскости. Маска, используемая для ограничения графических действий, так чтобы они оказывали воздействие только на некое подмножество набора битов цвета.
- **point source**
Точечный источник. Используемый как источник какого-то события *набор прямоугольников (rectangle set)*, состоящий из одной точки.
- **pointer**
Указатель мыши. Некий объект на экране, отслеживающий позицию устройства указания (напр., мыши, планшета, трекбола или джойстика). Photon имеет несколько изображений указателя мыши, указывающих на различные состояния: Основное, Занят, Помощь, Перемещение, Изменение размера, Двухтавр, Блокировка ввода.
- **Press-drag-release (PDR)**
Нажать-тащить-бросать. Метод выбора пункта меню путём нажатия кнопки мыши в момент, когда указатель мыши указывает на кнопку меню, перетаскивания указателя до тех пор, пока не высветится нужный пункт, и отпускания кнопки мыши.
- **print context**
Контекст печати. *Контекст прорисовки (draw context)*, в котором события прорисовки Photon'a направляются в файл, в противоположность направлению на экран (принимаемый по умолчанию контекст прорисовки) или в память (*контекст памяти (memory context)*).
- **printer driver**
Драйвер принтера. Программа, преобразующая формат потока прорисовки Photon'a в формат, пригодный для принтера, включая PostScript, Hewlett-Packard PCL, и Canon.
- **procreated widget**
Порождённый виджет. Виджет, созданный другим виджетом (а не приложением), такой как, например, PhList или PtText, созданный виджетом PtComboBox. Также называется *подчинённый потомок (subordinate child)*.

- **pterm**
Консоль Photon'a. Терминал Photon'a – приложение, которое в окне Photon'a эмулирует алфавитно-цифровой терминал.
- **pulse**
Импульс. Малое сообщение, не требующее отклика, используется для асинхронной передачи сообщений в приложениях Photon'a.
- **pv**
Просмотрщик рисунков. См. *просмотрщик рисунков (Image Viewer)*.
- **PWM**
Оконный менеджер Photon'a. См. *Оконный Менеджер Photon'a (Photon Window Manager)*.
- **raw event**
Неотфильтрованное событие. Некое событие ввода, которое не назначено какой-то локации в пространстве событий Photon'a. Также называется *несфокусированным событием (unfocused event)*.
- **raw callback**
Неотфильтрованная ответная реакция. Функция, позволяющая приложению отзываться непосредственно на события Photon'a, такие как события перетаскивания. Также называется *обработчиком событий (event handler)*.
- **realize**
Реализация. Вывод на экран виджета и его потомков, возможно делая их интерактивными.
- **rectangle set**
Набор прямоугольников. Массив неперекрывающихся прямоугольников, связанных с каким-то событием.
- **region**
Регион. Прямоугольная область в пространстве событий Photon'a, используемая приложением для сбора и генерирования событий.
- **resize policy**
Политика изменения размеров. Правила, управляющие тем, как виджет изменяет свои размеры, когда изменяется его содержание.
- **resource**
Ресурс. Некий атрибут виджета, такой как цвет заполнения, размеры или список ответных реакций.
- **root region**
Корневой регион. Самый задний регион пространства событий Photon'a.
- **sensitive**
Чувствительность. Состояние региона по отношению к событиям. Если регион является *чувствительным (sensitive)* к определённому типу событий, владелец региона накапливает копии всех тех событий, которые интересуют регион.
- **setup function**
Установочная функция. Функция, вызываемая после создания модуля PhAB.
- **shelf**
"Полка". Приложение, которое прикрепляет к внешнему краю экрана свои области. Вы можете добавить плагины, чтобы настроить эти области – такие плагины, как панель задач, плагин запуска, часы, "лупу".
- **Snapshot**
"Снимок". Приложение Photon'a для "захвата" образов с экрана.
- **specific placement**
Определённое месторасположение. Месторасположение региона, когда у него определены один или более братьев. Является противоположностью *месторасположения по умолчанию (default placement)*.

- **subordinate child**
Подчинённый потомок. Виджет, созданный другим виджетом (а не приложением), такой как PtList и PtText, созданные виджетом PtComboBox. Также известен как *порождённый виджет (procreated widget)*.
- **table-of-contents (TOC) file**
Файл таблицы содержания (ТОС-файл). В *Просмотрщике Помощи (Helpviewer)* Photon'a – файл, описывающий иерархию тем помощи.
- **taskbar**
Панель задач. Плагин "полки", отображающий иконки, представляющие из себя приложения, выполняющиеся в настоящий момент.
- **tile**
"Черепица". Структура данных, используемая для построения связного списка прямоугольников, такого как список повреждённых частей интерфейса.
- **topic path**
Путь к теме. Информация помощи, определяемая строчкой заголовков, отделённых слэшами (знаком "/").
- **topic root**
Корень темы. Путь к теме, используемый как начальная точка отыскания тем помощи.
- **topic tree**
Дерево тем. Иерархия информации помощи.

- **translation file**
Файл перевода. Файл, содержащий строки для приложения PhAB. Имеется один файл перевода для каждого языка, поддерживаемого приложением.
- **unfocused event**
Несфокусированное событие. См. *неотфильтрованное событие (raw event)*.
- **Unicode**
Уникод. 16-битовая схема кодирования по стандарту ISO/IEC 10646 для представления символов, используемых в большинстве языков.
- **UTF-8**
Кодирование символов по *Уникоду (Unicode)*, где каждый символ представлен одним, двумя или тремя байтами.
- **widget**
Виджет. Компонент (напр., кнопка) графического пользовательского интерфейса.
- **widget class**
Класс виджета. Некий шаблон для виджетов, которые выполняют схожие функции и предоставляют один и тот же общедоступный интерфейс. Например, PtButton является классом виджета.
- **widget database**
База данных виджетов. В PhAB'e – модуль, содержащий виджеты, которые могут быть в любой момент скопированы в окно, диалог или иной контейнер.
- **widget family**
Семейство виджетов. Иерархия *экземпляров (instances)* виджетов. Например, окно и виджеты, в нём содержащиеся.
- **widget instance**
Экземпляр виджета. См. *экземпляр (instance)*.

- **window frame region**
Регион рамки окна. Регион, который PWM добавляет к окну. Он позволяет Вам перемещать, изменять размеры, сворачивать в иконку и закрывать окно.
- **Window Manager**
Оконный менеджер. См. *Оконный Менеджер Photon'a (Photon Window Manager)*.
- **window module**
Модуль окна. Модуль PhAB'a, обрабатываемый как экземпляр виджета PtWindow.
- **window region**
Регион окна. Регион, соответствующий окну приложения.
- **work procedure**
Рабочая процедура. Функция, которая вызывается, когда у приложения нет необработанных ("висящих") событий Photon'a.
- **workspace**
Рабочее пространство. См. *консоль (console)*.
- **workspace menu**
Меню рабочего пространства. Конфигурируемое меню, отображающееся, когда Вы нажимаете или щёлкаете правой кнопкой мыши в момент, когда указатель мыши находится на заднем плане рабочего стола.

Ненеобходимое послесловие переводчика.

Уф-ф-ф...

Теперь, завершив перевод и уже (параллельно с переводом) побарахтавшись в PhAB'e, вижу, что всё надо бы переписать заново, с нуля – с правильной терминологией и точным описанием. В общем, типичная реакция программера на уже сделанный продукт.

Но, и как тоже типично для типичного программиста, эта бета-версия так и пойдёт в свет в качестве релиза... Узнаваемо, не правда ли? [*так ведь и пошла :0) – прим. редактора*]

Тем, кто возьмётся портировать это на нормальный русский язык и отловить множество багов, выражаю заранее свою благодарность и просьбу вернуть обратно отлаженную версию.

В заключение выражаю благодарность Майе Зайцевой, осуществившей компьютерный набор этого документа и проявившей бесконечное терпение и недюжинную проницательность при чтении моего рукописного текста.

С уважением, Зайцев В. aka ZZZ

2 октября 2003г.